Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Benjamin Kastelic

Razširitev metode za iskanje ujemanj med podatkovnimi shemami s podporo za dodatne podatkovne tipe in upoštevanjem strukture podatkovne sheme

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2015

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Benjamin Kastelic, z vpisno številko **63090024**, sem avtor magistrskega dela z naslovom:

Razširitev metode za iskanje ujemanj med podatkovnimi shemami s podporo za dodatne podatkovne tipe in upoštevanjem strukture podatkovne sheme

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom prof. dr. Matjaža Branka Juriča,
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

V Ljubljani, 8. aprila 2015 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Matjažu Branku Juriču za pomoč pri izdelavi, in dr. Sebastijanu Šprangerju za nasvete pri oblikovanju magistrske naloge. Zahvaljujem se tudi Moni Radež, ki je opravila lektoriranje celotne naloge. Zahvaljujem se še svoji celotni družini, ki je tekom izdelave naloge prenašala moje nerganje.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Pristopi za iskanje ujemanj med podatkovnimi shemami **5**

2.1 Osnovni pojmi 6

2.2 Klasifikacija pristopov za iskanje ujemanj 8

2.3 Iskanje ujemanj na osnovi shem 10

2.3.1 Lingvistični pristopi 10

2.3.2 Pristopi na podlagi omejitev elementov 12

2.3.3 Pristopi na podlagi strukture sheme 13

2.4 Iskanje ujemanj na osnovi instanc 14

2.5 Iskanje ujemanj na podlagi ponovno uporabljenih informacij . 15

2.6 Iskanje ujemanj z združevanjem pristopov 15

2.6.1 Hibridni pristopi 16

2.6.2 Kompozitni pristopi 17

3 Osnovni koncepti genetskih algoritmov **19**

3.1 Operacije genetskih algoritmov 20

3.2 Generacijski evolucijski algoritem 24

4 Metoda za iskanje ujemanj s pomočjo genetskih algoritmov **29**

4.1 Opis delovanja metode 29

4.1.1 Modeliranje problema 30
4.1.2 Prilagoditve evolucijskega algoritma 32
4.1.3 Omejitve evolucijskega procesa 36
4.1.4 Strategije za ocenjevanje podobnosti 38

5 Izboljšana metoda **43**
5.1 Identifikacija možnih izboljšav 43
5.2 Izboljšave 44
5.2.1 Uporaba podatkov sheme 44
5.2.2 Podpora za dodatne podatkovne tipe 46
5.3 Implementacija 48

6 Vrednotenje in primerjava predlagane metode **51**
6.1 Eksperimentalni protokol in parametri 52
6.2 Izvedba testiranja in rezultati 54
6.2.1 Primerjava z originalno metodo 54
6.2.2 Iskanje kompleksnih preslikav 58
6.2.3 Večje število zapisov 60
6.2.4 Časovna zahtevnost 61

7 Sklep **67**

Slike

2.1 Primer sheme SQL. 6
2.2 Števnosti lokalnih preslikav. 7
2.3 Delitev pristopov za iskanje ujemanj. 9
2.4 Primer uporabe informacij predhodno najdenih ujemanj. . . . 16
2.5 Primerjava hibridnega in kompozitnega pristopa. 18

3.1 Primer sheme, predstavljene z drevesno strukturo. 20
3.2 Primer operacije razmnoževanja. 21
3.3 Primer operacije križanja. 22
3.4 Primer operacije mutacije. 23

4.1 Primer enostavne preslikave med shemama. 32
4.2 Primer instanc dveh shem in preslikav med njima. 32
4.3 Postopek križanja. 35
4.4 Primer nesmiselne preslikave. 37

5.1 Primer kategoriziranih podatkov. 44

6.1 Primer kompleksne preslikave numeričnega tipa. 60
6.2 Primer kompleksne preslikave z različnimi podatkovnimi tipi. . 61
6.3 Čas iskanja ujemanja in ocena uspešnosti skozi generacije pri uporabi entitetno orientirane strategije. 63
6.4 Čas iskanja ujemanja in ocena uspešnosti skozi generacije pri uporabi vrednostno orientirane strategije. 64

6.5 Čas iskanja ujemanja in ocena uspešnosti skozi generacije pri uporabi vrednostno orientirane strategije pri 1000 zapisih. . . 65

Tabele

4.1 Nabor možnih operacij nad elementi sheme. 31

6.1 Statistika elementov in preslikav v uporabljenih shemah. . . . 54
6.2 Vrednosti parametrov, uporabljenih pri testiranju. 55
6.3 Rezultati testiranja – delno prekrivanje podatkov. 56
6.4 Rezultati testiranja – popolno prekrivanje podatkov. 57
6.5 Rezultati testiranja – brez prekrivanja podatkov. 59

Seznam uporabljenih kratic

kratica	slovensko	angleško
GA	genetski algoritem	genetic algorithm
GP	genetsko programiranje	genetic programming
KDE	ocenjevanje funkcij gostote verjetnosti	kernel density estimation
NLP	procesiranje naravnega jezika	natural language processing
SQL	strukturirani povpraševalni jezik	structured query language
XML	razširljiv označevalni jezik	extensible markup language

Povzetek

Z iskanjem ujemanj med podatkovnimi shemami želimo odkriti čim več semantično enakovrednih elementov med dvema shemama in določiti povezave med njimi. Ta proces je ena izmed glavnih aktivnosti pri integraciji podatkov. Večina obstoječih metod za iskanje ujemanj med shemami ima še vedno težave s kompleksnimi preslikavami. Prav zato smo se odločili, da izboljšamo eno od obstoječih metod za iskanje ujemanj med shemami, ki primarno rešuje ta problem. Izbrana metoda temelji na evolucijskem algoritmu, ki postopoma generira boljše posameznike (preslikave) samo na podlagi podatkovnih instanc. Ker se lahko zgodi, da v nekem scenariju podatkovne instance niso na voljo, smo v ta namen razvili izboljšano metodo, ki upošteva tudi podatke sheme. Metodo smo še dodatno razširili s podporo za dodatne podatkovne tipe. Izboljšano metodo smo ocenili na enakih testnih scenarijih kot originalno metodo. Ugotovili smo, da je naša izboljšana metoda v povprečju za 20 % bolj natančna od originalne pri iskanju tako enostavnih kot tudi kompleksnih preslikav.

Ključne besede: podatkovna integracija, iskanje ujemanj in preslikav med shemami, evolucijski algoritmi

Abstract

Schema matching aims at identifying semantically similar elements of two schemas and determining the mappings between them. This process is one of the main activities in data integration. Most of the existing methods for finding mappings between schemas still have difficulties with complex mappings. Therefore, we decided to improve one of the existing methods which deals primarily with finding complex mappings. The chosen method in based on an evolutionary algorithm that generates progressively better individuals (mappings) only on the basis of data instances. It may happen that in a certain scenario the data instances are not available. That is why we have developed an improved method which also takes the schema data into consideration. We have further improved the chosen method by adding support for additional data types. Our improved method was graded on the same test scenarios as the original method. We have found that our method is 20 % more accurate on average than the original one for both simple and complex mappings.

Keywords: data integration, schema mapping and matching, evolutionary algorithms

Poglavje 1

Uvod

Smo v času, ko je na trgu poslovnih aplikacij iz dneva v dan prisotnih več in več rešitev, ki večinoma rešujejo specifičen problem (npr. sistem za podporo poslovanju, za delo s strankami, za delo z naročili ...). Manjša podjetja in podjetja, ki so še v razvoju, velikokrat posegajo po takšnih aplikacijah, saj nimajo toliko kapitala, da bi si lahko privoščile nakup celostnih sistemov. Ker vsaka aplikacija pokriva drugačno področje, ima vsaka aplikacija praviloma tudi svojo bazo podatkov. Zato se pogosto dogaja, da se podatki, ki jih hranijo te aplikacije, vsaj deloma prekrivajo. Tu se pojavi vprašanje, ali ne bi bilo smiselno, da bi si aplikacije delile podatke. Ker je utopično pričakovati, da bi vsak proizvajalec dodal lastni aplikaciji podporo za integracijo z aplikacijami preostalih ponudnikov, je smiselno, da se razvije mehanizem, ki bi to počel avtomatsko. Iz te ideje se je razvilo področje integracije aplikacij, ki je danes zelo aktualno.

Kot smo omenili, je eno možno področje uporabe integracije aplikacij ravno integracija med aplikacijami znotraj enega podjetja. To je eno od klasičnih področij, kjer se je do zdaj izvajala integracija. S porastom oblačnih aplikacij (predvsem SaaS aplikacij) se tudi integracija aplikacij seli v oblak, kjer je tako lahko na voljo kot storitev, ki povezuje tako SaaS aplikacije kot tudi standardne aplikacije odjemalec - strežnik, ki živijo na strežnikih podjetij.

Proces integracije aplikacij pokriva več nivojev: integracijo uporabniških vmesnikov, poslovno integracijo, integracijo procesov in podatkovno integracijo. Tema naše magistrske naloge se dotika zadnjega nivoja – podatkovne integracije. Namen podatkovne integracije med aplikacijami je vsem udeleženim aplikacijam zagotavljati celovite in točne podatke. To v praksi pomeni, da je treba spremembe v eni aplikaciji propagirati do vseh ostalih aplikacij, in to karseda hitro. Največji izziv pri reševanju tega problema predstavljajo velike razlike v podatkovnih shemah med aplikacijami. Da je podatek iz ene aplikacije uporaben v drugi, je treba podatek ustrezno preoblikovati v obliko, ki je primerna za drugo aplikacijo. Preden pa lahko določimo ustrezne transformacije med podatkovnimi shemami, moramo najprej odkriti podobnosti oziroma ujemanja med njimi.

V naši magistrski nalogi se bomo posvetili metodi za iskanje ujemanj med podatkovnimi shemami, ki deluje na principu genetskih algoritmov. Razlog za izbiro te metode iz velike množice obstoječih metod je ravno v evolucijskem pristopu reševanja problema, ki je sposoben iskati tako enostavna kot kompleksna ujemanja med shemami. Metoda deluje tako, da iz začetne populacije ujemanj skozi iteracije razvija čedalje boljša ujemanja med shemami, na podlagi katerih se lahko generirajo ustrezne transformacije med njimi. Ker metoda ujemanja išče na podlagi podatkovnih instanc, lahko pride do scenarija, ko podatki ali sploh niso na voljo ali pa so zelo pomanjkljivi. V takem primeru bo metoda dajala slabe rezultate. Zato predlagamo izboljšavo metode na način, da jo dopolnimo v hibridno metodo, ki bo ujemanja med shemami iskala tudi na podlagi informacij, ki so prisotne v shemah. Ker pa izbrana metoda deluje samo za podatke tekstovnega tipa, jo bomo dodatno dopolnili tako, da bo z njo možno iskati preslikave tudi med ostalimi tipi podatkov.

V drugem poglavju si bomo ogledali obstoječe pristope in metode za iskanje ujemanj med podatkovnimi shemami in jih primerjali. V tretjem poglavju predstavimo značilnosti genetskih algoritmov, na katerih temelji izbrana metoda za iskanje ujemanj. V četrtem poglavju predstavimo metodo iz [1], ki

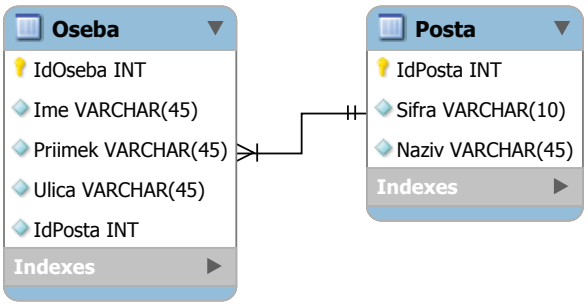
predstavlja osnovo te naloge. Omenimo tudi njene slabosti, ki jih v petem poglavju poskušamo izboljšati oziroma odpraviti. Tu podamo predloge za izboljšave in predstavimo dve izbrani izboljšavi. V šestem poglavju izvedemo testiranje izboljšane metode in rezultate primerjamo z originalno metodo. V sklopu tega poglavja opravimo še dodatne teste za iskanje kompleksnih preslikav in pa test iskanja preslikav nad večjim naborom podatkov. Opravimo pa tudi merjenje časovne zahtevnosti metode pri različnih scenarijih. Na koncu, v sedmem poglavju, predstavimo ugotovitve, sklepe in smernice za nadaljnje delo.

Poglavje 2

Pristopi za iskanje ujemanj med podatkovnimi shemami

Iskanje ujemanj med podatkovnimi shemami je uporabno na mnogih področjih. Najbolj izstopa področje podatkovnih baz, kjer se z iskanjem ujemanj poskuša rešiti problem sinhronizacije in integracije podatkov med bazami. V praksi se to največkrat dogaja v velikih podjetjih ali organizacijah, kjer je v uporabi veliko zalednih aplikacij, ki podpirajo poslovne procese. Veliko teh aplikacij ponavadi hrani zelo podobne podatke in zato je dobro, da obstaja mehanizem, ki skrbi za sinhronizacijo teh podatkov med vsemi aplikacijami.

Iskanje ujemanj med podatkovnimi shemami je zelo zahteven proces. Glavni razlog za to je ogromno število možnih načinov, kako lahko predstavimo in poimenujemo podatke, ki jih hrani podatkovna shema. Zato ne moremo pričakovati, da bi to izvajali ročno. Veliko raziskovalnih področij, kot so integracija in transformacija podatkovnih shem, strojno učenje, pridobivanje informacij itd., si prizadeva, da se ta proces v veliki meri avtomatizira. Glavni cilj tega poglavja je predstavitev pristopov, ki se uporabljajo pri procesu iskanja ujemanj, in da razložimo njihove glavne lastnosti ter področja uporabe.



Slika 2.1: Primer sheme SQL z dvema tabelama.

2.1 Osnovni pojmi

Preden nadaljujemo predstavitev pristopov, si najprej oglejmo nekaj osnovnih pojmov, povezanih z iskanjem ujemanj med podatkovnimi shemami. Poznavanje teh pojmov bo olajšalo razumevanje razlag, ki sledijo.

Podatkovna shema

S podatkovno shemo v formalni notaciji predstavimo entitete, ki so shranjene v podatkovnih repozitorijih. Struktura sheme določa, na kakšen način in iz kakšnih podatkov je sestavljena entiteta. To v podatkovni shemi določimo z elementi in hierarhijo med njimi. Vsakemu elementu sheme je možno definirati, kakšnega podatkovnega tipa je, kakšna je njegova zaloga vrednosti ter kakšne so njegove omejitve (npr. številnost, obveznost podatka, privzeta vrednost ...). Najbolj znani predstavniki podatkovnih shem so sheme XML ali SQL (slika 2.1). V shemah XML elemente predstavljajo elementi XML, v shemah SQL pa tabele in stolpci.

Ujemanje shem

Iskanje ujemanj med shemami je proces iskanja elementov, ki predstavljajo podobno semantično vsebino v shemi. Če med shemami obstajajo ujemanja, lahko na podlagi tega generiramo *preslikave*, ki nam omogočajo transforma-

Številnost	Elementi S1	Elementi S2	Preslikava
1:1	Cena	Vrednost	Cena = Vrednost
N:1	Ulica, HišnaŠt	Naslov	CONCAT(Ulica, HišnaŠt) = Naslov
1:N	Naslov	Ulica, HišnaŠt	SPLIT(Naslov) = {Ulica, HišnaŠt}
N:M	N.ŠtNaročila, N.ŠtArtikla, A.ŠtArtikla, A.ImeArtikla	O.Naročilo, O.Artikel	SELECT N.ŠtNaročila, A.ImeArtikla FROM N, A WHERE N.ŠtArtikla = A.ŠtArtikla = {O.Naročilo, O.Artikel}

Slika 2.2: Številnosti lokalnih preslikav.

cijo podatkov iz ene sheme v drugo.

Preslikave

Rezultat operacije iskanja ujemanj med shemama S1 in S2 je množica preslikav med njunimi elementi. Kot primer vzemimo shemi S1 in S2, kjer prva vsebuje elementa *Ulica* in *HisnaŠt*, druga pa element *Naslov*. Vidimo, da obstaja ujemanje med obema shemama, ker elementi sheme S1 in S2 hranijo isto semantično vsebino. S pomočjo operacij `CONCAT` in `SPLIT` za združevanje in ločevanje nizov lahko iz ujemanja izpeljemo naslednji preslikavi:

$$\texttt{CONCAT}(\text{S1}.\mathit{Ulica}, \text{S1}.\mathit{HisnaŠt}) \leftrightarrow \text{S2}.\mathit{Naslov}$$
$$\{\text{S1}.\mathit{Ulica}, \text{S1}.\mathit{HisnaŠt}\} \leftrightarrow \texttt{SPLIT}(\text{S2}.\mathit{Naslov})$$

Številnosti preslikav

Element sheme S1 ali S2 je lahko del nobene, ene ali več preslikav. Poleg tega se lahko znotraj preslikave eden ali več elementov sheme S1 slika v enega ali več elementov sheme S2. Iz tega sledi, da poznamo naslednje vrste preslikav: *enostavne* preslikave številnosti 1:1 in *kompleksne* preslikave številnosti 1:N, N:1 in N:M. Iskanje preslikav na nivoju elementov je tipično omejeno samo na generiranje preslikav številnosti 1:1, 1:N in N:1, medtem ko je potrebno za generiranje preslikav številnosti N:M upoštevati celotno strukturo in relacije elementov v shemi. To imenujemo iskanje preslikav na nivoju sheme.

Slika 2.2 prikazuje tipe lokalnih številnosti na primeru štirih preslikav. V prvi vrstici je prikazana preslikava številnosti 1:1, ki označuje ekvivalenco med elementoma *Cena* in *Vrednost*. Kadar iščemo ujemanje med več elementi shem, potrebujemo izraz, s katerim povemo, kako so elementi shem povezani. V drugi in tretji vrstici vidimo, da se elementa *HišnaŠt* in *Ulica* iz sheme S1 slikata v element *Naslov* sheme S2 in obratno. To je na preslikavi v drugi vrstici prikazano s funkcijo `CONCAT` in s funkcijo `SPLIT` v tretji vrstici. V zadnji vrstici je predstavljen primer preslikave številnosti N:M. Za opis preslikave je uporabljen stavek SQL, ki združuje elemente iz tabel N in A sheme S1 v tabelo O sheme S2.

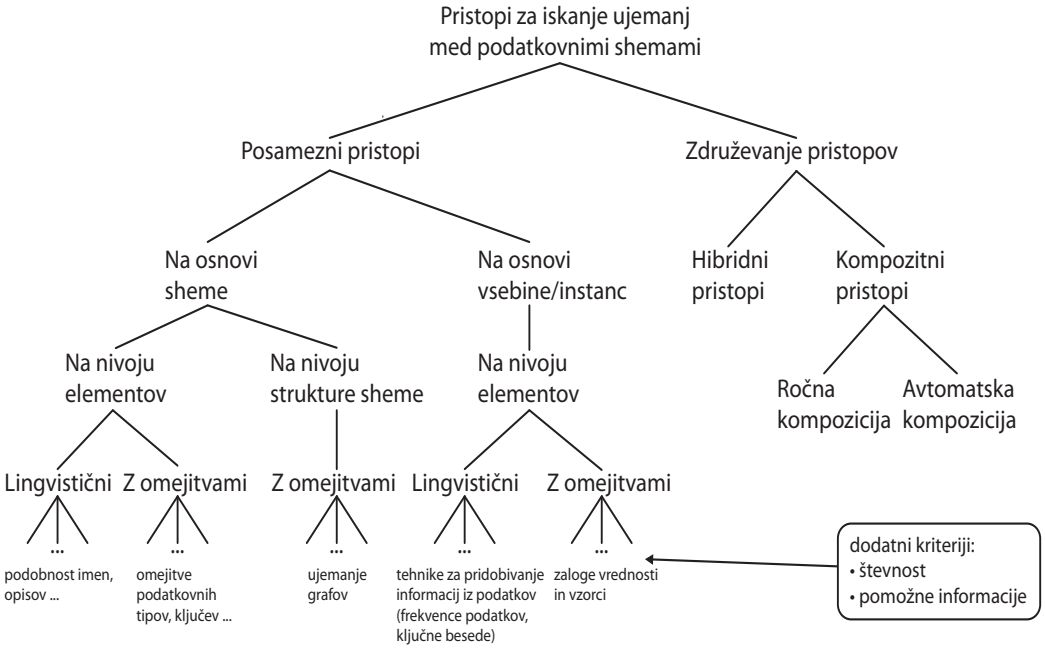
Globalni primeri številnosti glede na celoten rezultat iskanja ujemanj (s tem mislimo vse najdene preslikave) so v veliki meri ortogonalni na posamezne preslikave. Kot primer poglejmo preslikavo iz prve vrstice tabele 2.2. Številnost preslikave je globalna, če ni nobene druge preslikave iz S2, ki se slika v S1.*Cena*, in nobene druge preslikave iz S1, ki se slika v S2.*Vrednost*. Po drugi strani, če za element S1.*Cena* obstaja poleg preslikave v S2.*Vrednost* še kakšna druga preslikava, imamo globalno številnost 1:N v kombinaciji z lokalnimi preslikavami številnosti 1:1 ali 1:N [3].

2.2 Klasifikacija pristopov za iskanje ujemanj

Pristopi za iskanje ujemanj med podatkovnimi shemami se med seboj ločijo po treh lastnostih [4, 29, 13]:

- kakšne informacije izkoriščajo za iskanje ujemanj,
- na kakšen način te informacije procesirajo in
- kako predstavijo najdena ujemanja.

Kot je razvidno iz slike 2.3, ločimo naslednje glavne skupine kriterijev za delitev pristopov za iskanje ujemanj [3, 28]:



Slika 2.3: Delitev pristopov za iskanje ujemanj med podatkovnimi shemami.

- **Uporaba sheme ali podatkovnih instanc:** pristopi za iskanje ujemanj uporabljajo informacije, pridobljene iz sheme (npr. imena elementov, opise, podatkovne tipe), lahko pa za iskanje ujemanj izkoriščajo podatkovne instance.

- **Delovanje na nivoju enega elementa ali strukture elementov:** za iskanje ujemanj se izvajajo primerjave na nivoju elementov (atributov) ali se primerjajo skupine elementov, ki so del strukture.

- **Izvajanje primerjav:** lingvistični pristopi primerjajo imena in opise elementov, obstajajo pa tudi pristopi, ki delujejo na osnovi omejitev elementov v shemi.

- **Uporaba dodatnih informacij:** pristopi za iskanje ujemanj uporabljajo dodatne informacije iz različnih virov, kot so razni slovarji, podatki o predhodno najdenih ujemanjih ...

- **Združevanje pristopov:** za doseganje boljših rezultatov lahko obstoječe pristope kombiniramo med seboj. Z združevanjem različnih pristopov v povezano celoto dobimo hibridne pristope, z združevanjem rezultatov posameznih pristopov pa kompozitne pristope.
- **Številnost preslikav:** v preslikavi je lahko en ali več elementov prve sheme povezan z enim ali več elementi druge sheme (npr. 1:1, N:1, 1:N, N:M).

Večina kriterijev za delitev pristopov se nanaša na informacije, ki pristopom služijo kot vhodni podatki. Delitvi pristopov glede na način procesiranja vhodnih informacij (npr. strojno učenje ali iskanje podobnosti med nizi) se izogibamo, saj so ti pristopi v veliki meri odvisni prav od tipa in karakteristik vhodnih podatkov. Prav tako ne razlikujemo pristopov glede na tipe shem (npr. SQL in XML), med katerimi iščejo ujemanja, saj za samo delovanje metode to ni pomembno – važna je predstavitev podatkov sheme.

2.3 Iskanje ujemanj na osnovi shem

Pristopi za iskanje ujemanj na osnovi shem uporabljajo samo informacije, ki so na voljo v sami shemi [6, 30]. Informacije, ki jih lahko pridobimo iz sheme, so v precejšnji meri odvisne od tipa sheme. Primeri takšnih informacij so: imena elementov, opisi, podatkovni tipi, zaloge vrednosti, omejitve ... Najprej bomo predstavili dva tipična predstavnika pristopov, ki delujeta na nivoju enega elementa: lingvistični pristop in pristop za iskanje ujemanj na osnovi omejitev. V nadaljevanju si bomo ogledali še pristope, ki iščejo ujemanja med strukturami elementov.

2.3.1 Lingvistični pristopi

Lingvistični pristopi izkoriščajo tekstovne informacije elementov sheme, kot so imena in opisi. Imena elementov so najosnovnejši sestavni del sheme in predstavljajo prvi nivo informacij za ugotavljanje podobnosti med elementi.

Podobnost med imeni lahko ugotavljamo *sintaktično* s primerjanjem zapisa imen ali pa *semantično* s primerjanjem pomenov imen elementov.

S *sintaktičnim* načinom ocenjevanja, podobnosti med imeni elementov izračunamo s primerjanjem dveh besed samo na podlagi njunega zapisa. Primer najenostavnejše metode za izračun podobnosti je ugotavljanje popolne identičnosti dveh besed. V primeru shem XML je ta metoda povsem zadovoljiva, saj obstaja za elemente iz istih imenskih prostorov velika verjetnost, da predstavljajo isto vsebino. V vseh ostalih primerih pa je bolje uporabiti načine za "približno" ocenjevanje podobnosti, ki jo izračunamo na podlagi skupnih nizov, urejevalne razdalje, izgovorjave in soundex kodiranja [8] ...

Po drugi strani pa *semantični* način ocenjevanja podobnosti primerja imena elementov glede na njihove terminološke lastnosti – ali so imena sopomenke, nadpomenke ali enakozvočnice (npr. besedi *avto* in *avtomobil* sta sopomenki in sta zato podobni). Uporaba nadpomenk in sopomenk pri iskanju podobnosti med imeni zahteva uporabo tezavrov[1] in dodatnih slovarjev. Pri tem si lahko pomagamo s slovarji naravnih jezikov. V primerih, kadar imamo opravka s shemami, ki uporabljajo različne jezike, lahko uporabimo tudi večjezikovne slovarje (npr. angleško-nemški). Dodatno lahko iskalne metode uporabljajo slovarje, ki so specifični za neko domeno. Takšni slovarji vsebujejo sopomenke, skupne opise elementov, imena, okrajšave ... Pripravi takega slovarja je treba nameniti nekaj več truda, saj je pri gradnji treba paziti na konsistentnost.

Iskanje ujemanj med elementi na osnovi njihovih imen je možno izvajati na več nivojih sheme. En primer tega je, da se pri iskanju ujemanj upoštevajo tudi imena nadrejenih elementov. Na tak način lahko ugotovimo, da se element *avtor.ime* slika v element *ImeAvtorja*.

Poleg tega ta pristop ni omejen samo na iskanje enostavnih preslikav številnosti 1:1. Za izbrani element sheme lahko najdemo več preslikav (lahko najdemo ujemanje elementa *telefon* z elementoma *mobilni telefon* in *domači*

[1] Tezaver ali tezavrus je zbirka sopomenk, v kateri lahko najdemo besede s podobnim ali istim pomenom, včasih pa tudi protipomenke.

telefon).

Včasih se zgodi, da sheme poleg imen vsebujejo tudi opise posameznih elementov. Z lingvističnimi metodami lahko take opise analiziramo in na tak način določimo morebitne kandidate za preslikavo. Oglejmo si naslednja dva elementa:

S1: KZZSt //številka kartice zdravstvenega zavarovanja
S2: StZdravZav // št. zdravstvenega zavarovanja

Če so opisi elementov identični, lahko s precejšnjo gotovostjo trdimo, da se taki elementi ujemajo. To pa se v praksi le poredkoma zgodi. Zato je treba poseči po neke vrste lingvistični analizi za primerjavo teh opisov. V najenostavnejšem primeru gre za ekstrakcijo ključnih besed iz opisov, katere uporabimo pri iskanju ujemanja na podoben način, kot da bi primerjali imena elementov. Bolj kompleksne metode se poslužujejo procesiranja naravnega jezika. Z njimi se išče semantično podobne besedne zveze.

2.3.2 Pristopi na podlagi omejitev elementov

Sheme po večini definirajo omejitve za elemente, kot so tipi podatkov, zaloge vrednosti, izbirnost (angl. *optionality*), enoličnost (angl. *uniqueness*), številnost, relacije ... Če so tovrstne informacije prisotne v obeh vhodnih shemah, to lahko uporabimo kot osnovo za določanje podobnosti med njunimi elementi. Podobnost lahko določimo na osnovi podobnosti podatkovnih tipov, zalog vrednosti, značilnosti ključev (primarni, enolični, tuji), številnosti povezav itd.

Zaradi različnega poimenovanja omejitev z istimi karakteristikami je nujno, da se pri tem pristopu uporablja slovar, ki vsebuje ocene podobnosti za različne omejitve. Kot primer si lahko predstavljamo slovar, ki vsebuje ocene podobnosti za podatkovne tipe, ki izvirajo iz različnih jezikov za definicijo shem (npr. `string` in `varchar`).

Pristopi za iskanje ujemanj na osnovi omejitev težko določijo točne preslikave, saj se v shemah ponavadi nahaja več elementov s kompatibilnimi ome-

jitvami. Če upoštevamo samo omejitve podatkovnih tipov bodo vsi elementi z istim podatkovnim tipom označeni kot podobni. Kljub tej pomanjkljivosti pa je ta pristop še vedno uporaben pri omejevanju števila kombinacij elementov, ki jih je treba pregledati in je zato večinoma uporabljen v kombinaciji z drugimi pristopi (npr. iskanje na podlagi imen). S takimi kombinacijami iskalnikov nato dobimo bolj zanesljive rezultate.

2.3.3 Pristopi na podlagi strukture sheme

Pristopi za iskanje ujemanj na podlagi struktur shem izkoriščajo relacije med elementi. Tipi relacij, ki so nam na voljo, so odvisni od tipa sheme in jezika, s katerim je predstavljena. Relacije med elementi so ponavadi predstavljene z usmerjenimi ali neusmerjenimi grafi. Na podlagi takšnih informacij izvemo, kateri so tisti elementi, ki so podrejeni drugim elementom, kar dobimo preko medsebojnih povezav med elementi. Takšne omejitve lahko predstavimo kot strukture, na podlagi katerih je možno iskati ujemanja med njimi [17].

Kadar iščemo podobnost med hierarhično urejenimi strukturami, imamo za to na voljo dva pristopa: od zgoraj navzdol (angl. *top-down*) in od spodaj navzgor (angl. *bottom-up*). Za pristope od zgoraj navzdol ponavadi velja, da so manj potratni kot tisti od spodaj navzgor, ker so algoritmi zasnovani tako, da na višjih nivojih zmanjšajo število možnih primerjav, ki jih je potrebno izvesti na nižjih nivojih.

Slabost pristopa od zgoraj navzdol se pokaže, ko sta strukturi na najnižjem nivoju precej drugačni. To pomeni, da bo algoritem spregledal ujemanje med elementom *Naslov*, ki je na spodnjem nivoju sestavljen iz elementov *Ulica*, *PostnaŠt* in *Posta*, in elementom *Naslov*, ki vse informacije predstavi v eni besedni zvezi. Ravno nasprotno pa deluje pristop od spodaj navzgor: algoritem na spodnjem nivoju pregleda vse možne kombinacije preslikav med elementi in za razliko od prvega pristopa tako najde rešitve na tem nivoju, četudi se strukturi na najvišjem nivoju močno razlikujeta.

2.4 Iskanje ujemanj na osnovi instanc

Instance podatkov so eden najpomembnejših virov podatkov za iskanje ujemanj, saj lahko iz njih zelo dobro sklepamo, kateri elementi hranijo vsebinsko podobne podatke [26]. Instance podatkov so zelo uporabne v primerih, kadar so informacije o strukturi sheme nedostopne ali omejene. Metode na osnovi instanc podatkov se lahko lahko uporabijo za odkrivanje napačnih ali potrjevanje pravilnih preslikav, ki so bile pridobljene iz podatkov sheme. V primeru dvoumnih preslikav lahko določimo boljšo rešitev na podlagi podobnosti instanc. Večino pristopov, ki smo jih obravnavali pri pristopu na nivoju shem, lahko uporabimo tudi na nivoju instanc podatkov.

Pri tekstovnih podatkih se uporabljajo lingvistične metode. Za najboljše se izkažejo pristopi, ki temeljijo na iskanju ključnih besed ter na analizi frekvenc besed in besednih zvez. Z uporabo frekvenc besed lahko v bazi nekega podjetja razločimo med imeni oddelkov in imeni zaposlenih. Z iskanjem pojavitev posebnih besed in okrajšav je možno prepoznati elemente, ki vsebujejo geografska imena in naslove. Tako kot pri iskanju ujemanj na podlagi imen elementov se tudi pri tem pristopu podobnost med besedami izračuna s pomočjo različnih tehnik.

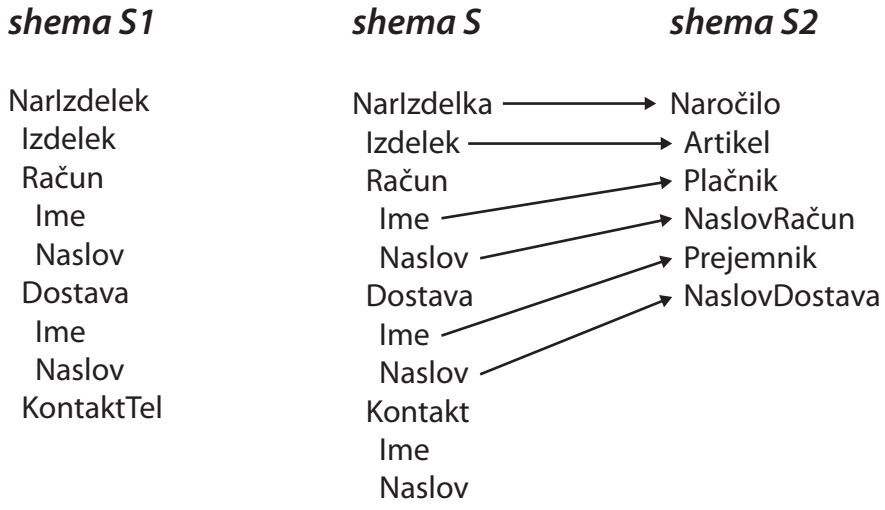
Za podatke, kot so nizi in števila, so v uporabi tehnike, ki delujejo na osnovi lastnosti omejitev. Primeri takšnih omejitev so povprečje in zaloga vrednosti števil, dolžina podatka, tip podatka, porazdelitve vrednosti elementov, frekvence znakov itd. To nam omogoča prepoznavanje telefonskih, poštnih in drugih številk, zemljepisnih imen, naslovov ... Pri uporabi takšnih tehnik je potrebno biti pazljiv, saj je možno, da element z omejitvijo za podatkovni tip `string` v resnici hrani števila v tekstovni obliki. V tem primeru bo iskalnik deloval napačno in vračal neustrezne preslikave.

2.5 Iskanje ujemanj na podlagi ponovno uporabljenih informacij

V prejšnjih poglavjih smo že omenili uporabo pomožnih virov informacij, kot so raznovrstni slovarji in pa druge informacije, ki jih posredujejo uporabniki sami. Še en možen pristop za izboljšanje učinkovitosti iskanja ujemanj je uporaba informacij o pogostih preslikavah, ki jih lahko pridobimo iz predhodno najdenih ujemanj. Takšen pristop je zelo obetaven, saj pričakujemo, da je veliko število podatkovnih shem znotraj neke domene precej podobnih (imajo podobne ali celo iste nazive elementov). Iz že najdenih ujemanj je možno ustvariti slovar preslikav, ki daje informacije o vseh možnih preslikavah za neki element. Uporaba takšnega pristopa je prikazana na sliki 2.4, kjer vidimo tri sheme: nad shemama S in S2 smo v preteklosti že našli ujemanje in tako že poznamo preslikave med njunimi elementi, želimo pa najti ujemanje med shemama S1 in S2. Ker sta si shemi S1 in S zelo podobni, lahko zelo enostavno najdemo preslikave med shemama S1 in S2. Velja pripomniti, da so takšni slovarji preslikav uporabni samo znotraj neke domene, saj ni nujno, da preslikava v eni domeni pomeni isto kot v drugi (npr. *plača* in *dohodek* sta si enakovredna v domeni za obračun plač, kar pa ne velja za domeno obračuna dohodnine).

2.6 Iskanje ujemanj z združevanjem pristopov

Uporabnost nekega pristopa je zelo odvisna od tipa informacij, ki jih izkorišča za iskanje ujemanj. Iskalnik ujemanj, ki uporablja samo en pristop za iskanje ujemanj, bo le stežka dosegal dobre rezultate na vseh vrstah shem in domen, v katerih te nastopajo. Zato je večina današnjih iskalnikov preslikav zgrajena na osnovi dveh ali več pristopov [7, 11]. To je izvedljivo na dva načina: na *hibridni* in *kompozitni* način. Hibridni iskalniki združujejo več pristopov na fiksen način, medtem ko kompozitni iskalniki združujejo rezultate samostojno



Slika 2.4: Primer uporabe informacij predhodno najdenih ujemanj.

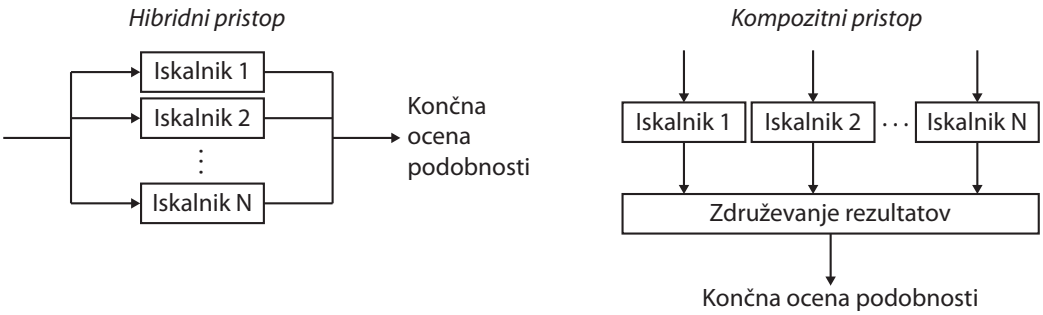
zagnanih iskalnikov, ki so lahko tudi hibridni ali kompozitni.

2.6.1 Hibridni pristopi

Hibridni iskalniki združujejo množico pristopov in na podlagi velike množice kriterijev in virov informacij (npr. podobnost imen, opisov, tipov podatkov), iščejo ujemanja med shemami. Tipično se pristopi znotraj iskalnika izvajajo zaporedno, kar pomeni, da se najprej identificira elemente s podobnimi podatkovnimi tipi in se šele nato preveri podobnost glede na imena elementov. Na tak način lahko odkrijejo boljše preslikave in dosegajo hitrejše izvajalne čase, kot če bi zaporedno izvajali posamezne metode. Prav tako so iskalniki take vrste bolj učinkoviti, saj lahko zaradi kombiniranja iskalnih kriterijev slabe kandidate za preslikave hitro izločimo iz procesa iskanja. Kot primer lahko pristop za iskanje preslikav na podlagi podobnosti struktur združimo s pristopom za iskanje ujemanj na podlagi podobnosti imen elementov. V takem primeru prvi pristop tipično uporabimo za identifikacijo približnih preslikav, slednjega pa za določitev končnih preslikav.

2.6.2 Kompozitni pristopi

Po drugi strani kompozitni pristopi združujejo rezultate več neodvisno izvedenih iskalnikov (tudi hibridnih ali kompozitnih). Združevanje pristopov na takšen način nam nudi veliko večjo fleksibilnost, kot če bi uporabili hibridne pristope, saj so ti precej bolj povezani, ker so del enega skupnega iskalnika. Prednost kompozitnih iskalnikov se izkaže pri poljubnem izbiranju nabora uporabljenih pristopov glede na posamezen primer. Pri tem velja upoštevati dejstvo, da so eni pristopi v določenih situacijah boljši kot drugi. Za kompozitne iskalnike velja nenapisano pravilo, da bi morali uporabniku omogočati poljubno nastavljanje načina, kako naj se vsebovani pristopi izvajajo; ali vzporedno ali zaporedno. Za slednje velja, da se rezultat prejšnje metode uporabi kot vhod za naslednjo metodo, s čimer dosežemo iterativno izboljševanje najdenih preslikav. Izbira metod in določanje vrstnega reda izvajanja izbranih metod se lahko zgodi avtomatsko, lahko pa takšne parametre določi uporabnik. Avtomatski način lahko znatno zmanjša število posegov, ki jih mora opraviti uporabnik, vendar je težko doseči takšno mero avtonomnosti prav za vsak možni primer. Kadar parametre določa uporabnik, je večja možnost, da bo iskalnik vračal boljše rezultate, kot če bi deloval avtomatsko, saj ima uporabnik ponavadi več znanja o domeni, znotraj katere išče ujemanja, in lahko s tem znanjem bolj natačno določi, katere metode so najbolj primerne za iskanje preslikav. Na koncu pa ima uporabnik tako ali tako zadnjo besedo, saj je namen iskalnika le predlaganje možnih preslikav, ki jih lahko uporabnik potrdi, jih popravi ali pa jih celo zavrže.



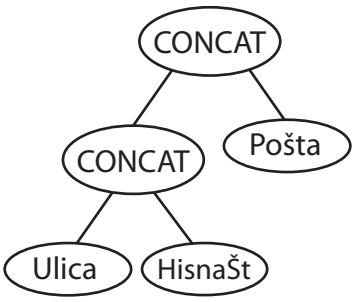
Slika 2.5: Primerjava hibridnega in kompozitnega pristopa.

Poglavje 3

Osnovni koncepti genetskih algoritmov

Evolucijski algoritmi temeljijo na principu naravnega procesa, ki vpliva na vse žive stvari – *naravna selekcija* [18]. GA so ena od najbolj poznanih tehnik evolucijskega programiranja. Koncept GA si je najlažje predstavljati kot hevristiko, ki deluje na principu genetskih operacij in naravne selekcije. Je neposredna evolucija algoritmov, katerih namen je induktivno učenje in ki se uporabljajo za reševanje optimizacijskih problemov. GA so tako kot druge evolucijske tehnike znani po tem, da znajo reševati tudi take probleme, kjer je možno več pravilnih rešitev. Naj tu povemo še, da obstaja tudi tehnika GP, ki deluje na podoben način kot GA. Razlika med obema tehnikama je v obliki podatkov, s katerimi operirata – pri GP so to programi, pri GA pa poljubne podatkovne strukture.

Metode GA in GP so prav tako znane po tem, da se odlično obnesejo pri iskanju rešitev v ogromnem – tudi neskončnem – iskalnem prostoru, kjer pa optimalna rešitev v mnogih primerih sploh ne obstaja, zato nam metodi kot rezultat vračata rešitve, ki so blizu optimalni.



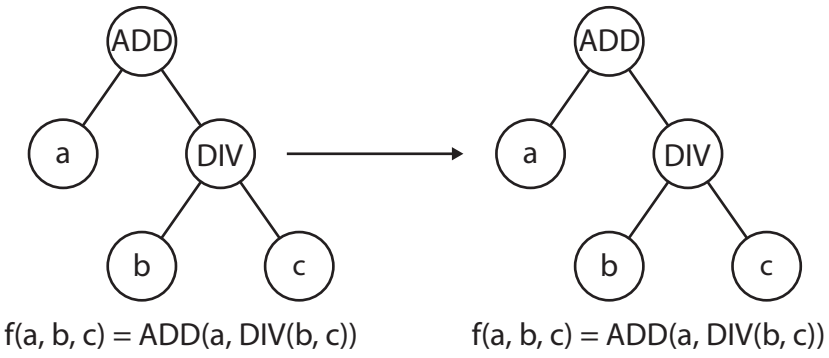
Slika 3.1: Primer sheme, predstavljene z drevesno strukturo.

3.1 Operacije genetskih algoritmov

GA operirajo s populacijo struktur, katerim pravimo posamezniki oziroma osebki (angl. *individuals*). Vsak osebek predstavlja možno rešitev danega problema. Če hočemo vpeljati metodo GA na problem iskanja preslikav, je treba rešitev predstaviti s primerno strukturo. V našem primeru je najboljši način uporaba drevesne podatkovne strukture, ki omogoča manipulacijo s strani GA [1]. V drevesu z vozlišči in povezavami med njimi, modeliramo kombinacije in soodvisnosti elementov sheme. Elementi sheme se nahajajo v listih drevesa, znotraj pa so vozlišča, ki določajo na kakšen način so listi (elementi) povezani med seboj. V terminologiji GA so listi znani kot terminali, vozlišča pa so funkcije, katerih namen je manipuliranje s terminali. Primer predstavitve sheme s tako strukturo je viden na sliki 3.1.

Med evolucijskim procesom se posameznike upravlja in spreminja s tremi glavnimi genetskimi operacijami [21]. To so **razmnoževanje** (angl. *reproduction*), **križanje** (angl. *crossover*) in **mutiranje** (angl. *mutation*). Operacije izvajamo na iterativni način, kjer z vsako iteracijo z neko verjetnostjo generiramo boljše posameznike.

Razmnoževanje je operacija, ki kopira posameznike iz prejšnje generacije v novo brez spreminjanja posameznikov (slika 3.2). Ta operacija deluje po načelu elitistične strategije, ki skrbi, da se genetska koda najboljših posameznikov obdrži skozi generacije. To pomeni, da če v zgodnejših generacijah odkrije dobrega posameznika, se bo tak posameznik prenašal v naslednje



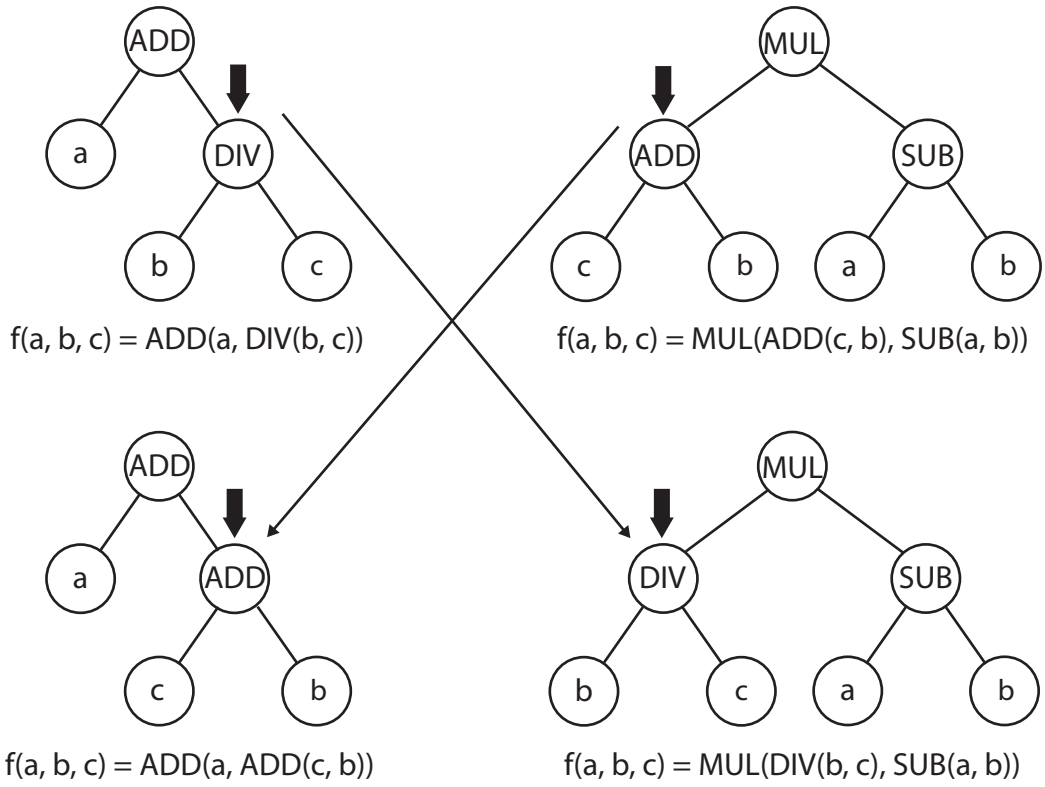
Slika 3.2: Primer operacije razmnoževanja.

generacije in na tak način prispeval k boljši skupni rešitvi problema.

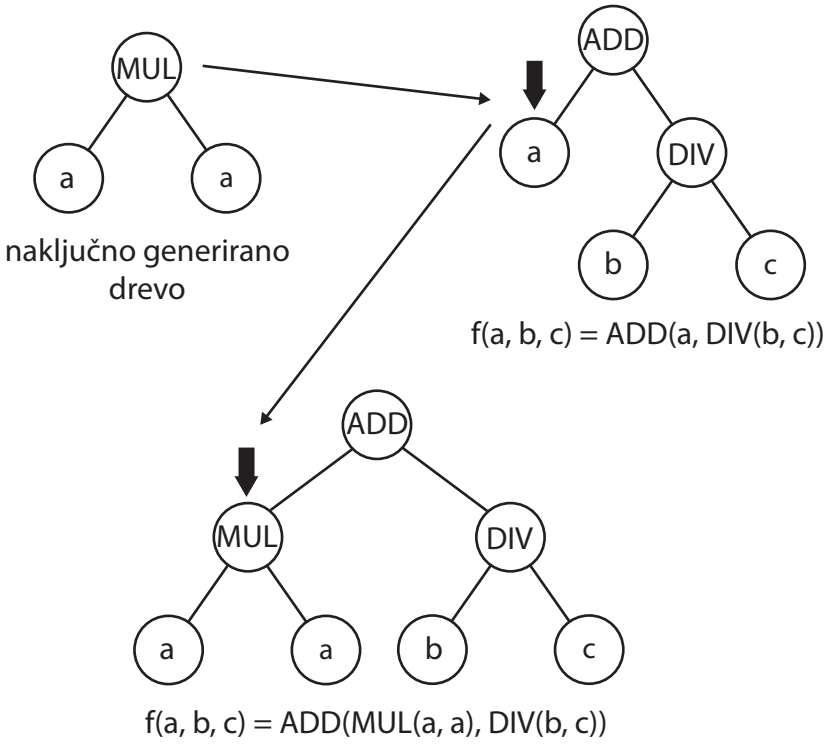
Operacija križanja omogoča, da se genetski material (npr. poddrevesa) izmenjuje med dvema posameznikoma. Takšna izmenjava nam generira dva ali več novih posameznikov, ki vsebujejo genetski material obeh staršev. Operacija poteka tako, da izbere dva starša iz celotne populacije, nato pa naključno izbere poddrevesa iz obeh staršev in jih združi. Tako dobimo nove posameznike oziroma otroke (slika 3.3).

Naloga operacije mutacije je, da ohranja minimalno raznolikost med posamezniki v populaciji, s čimer se izognemo prehitri konvergenci h končni rešitvi. Vsi posamezniki, ki jih dobimo z operacijo križanja, imajo enako verjetnost za izvedbo mutacije. Pri predstavitvi z drevesno strukturo to izvedemo tako, da naključno izberemo vozlišče (ali list) v drevesu in ga zamenjamo z naključno generiranim drevesom, kot je vidno na sliki 3.4.

Zelo pomembno je, da se vse operacije za vstavljanje in menjavo poddreves, ki jih izvajata križanje in mutacija, izvajajo z enakimi verjetnostmi. Tako imajo vsa vozlišča enako možnost za izbiro, kar zagotavlja raznolikost posameznikov znotraj populacije.



Slika 3.3: Primer operacije križanja.



Slika 3.4: Primer operacije mutacije.

3.2 Generacijski evolucijski algoritem

Evolucijski proces lahko izvajamo na dva načina: z uporabo linearnega (angl. *steady-state*) pristopa ali z uporabo generacijskega pristopa [19]. Glavna razlika med obema pristopoma je, da prvi ne uporablja generacij, medtem ko drugi uporablja natančno definirane in raznolike generacije. Metoda, ki jo predstavljamo, uporablja generacijski pristop, saj le ta najbolj zajame bistvo evolucije. Glavni koraki generacijskega pristopa so:

1. **Kreiranje začetne populacije:** Lahko jo generiramo naključno ali pa jo poda uporabnik. Velikost populacije (število posameznikov) v vsaki generaciji, tudi začetni, je parameter, ki ga je treba ustrezno nastaviti.

2. **Ocenjevanje posameznikov:** Vsakega posameznika v trenutni populaciji ocenimo glede na njegovo ustreznost kot rešitev danega problema.

   Ocenjevanje izvedemo na način, da vsakemu posamezniku priredimo vrednost, ki določa, kako ustrezna je rešitev problema, ki jo predstavlja ta posameznik. To storimo s funkcijo uspešnosti, ki boljšim rešitvam dodeli višje ocene, slabšim pa nižje.

3. **Preverjanje ustavitvenega pogoja:** Ustavitveni pogoj določa prag oziroma mejo, pri kateri se izvajanje algoritma ustavi. Tip pogoja je povsem odvisen od situacije. Največkrat uporabljeni pogoji so:

   - **število generacij:** maksimalno število generacij, ki jih lahko izvede algoritem,
   - **izvajalni čas:** po določenem času se izvajanje algoritma prekine,
   - **ocena populacije:** algoritem se ustavi, ko posameznik v populaciji doseže izbrano oceno ustreznosti,
   - **minimalna razlika med populacijami:** ko razlika med oceno ustreznosti prejšnje in trenutne populacije doseže določeno vrednost, se izvajanje ustavi zaradi predvidevanja, da se populacije v naslednjih generacijah ne bodo več pretirano spreminjale.

Če je ustavitveni pogoj izpolnjen, nadaljujemo s korakom 7.

4. **Reprodukcija:** Izbira najboljših N posameznikov, ki gredo neposredno v novo generacijo. Ta korak je opcijski in predstavlja tako imenovano *elitistično* strategijo.

5. **Izbor posameznikov:** Izbira M posameznikov, katerih potomci bodo sodelovali pri gradnji nove populacije.

   Po izvedbi vrednotenja populacije ima vsak posameznik dodeljeno oceno uspešnosti, ki določa, kako dober je posameznik. S pomočjo te ocene se lahko odločimo, kateri posamezniki so bolj primerni za vključitev v naslednje generacije. Strategije za izbiro segajo od najbolj preprostih do zelo kompleksnih – od izbire najboljših N posameznikov do naključnega izbiranja posameznikov glede na njihovo oceno uspešnosti. Najbolj pogosto uporabljene strategije za izbiro posameznikov so:

   - **ruleta**: verjetnost izbire posameznika je sorazmerna z oceno uspešnosti posameznika,
   - **turnir**: za vsako prosto mesto v naslednji generaciji se naključno izbere N posameznikov – v naslednjo generacijo gre tisti, ki ima najvišjo oceno uspešnosti,
   - **naključna**: posameznike se izbira povsem naključno,
   - **razvrstitev**: vseh N prostih mest v naslednji generaciji zapolni prvih najboljših N posameznikov,
   - **požrešna**: majhna skupina najboljših posameznikov ima nekoliko večjo verjetnost za napredovanje v naslednjo generacijo.

   Vsaka od strategij ima drugačen vpliv na raznolikost posameznikov v populaciji. Če je raznolikost majhna, bo algoritem počasneje konvergiral h končni rešitvi, če pa je prevelika, se lahko zgodi, da algoritem ne bo našel optimalne rešitve [20]. Izbira primerne strategije je povsem odvisna od tipa problema, ki ga rešujemo. Zato je povsem možno, da

se izbrana strategija v nekem primeru izkaže za dobro, spet v drugem primeru pa daje slabše rezultate.

6. **Križanje in mutiranje:** Nad izbranimi posamezniki izvedemo operaciji *križanja* in *mutacije*. Na ta način dobljene potomce skupaj z morebitnimi izbranimi elitnimi posamezniki združimo v novo populacijo in nadaljujemo s korakom 2.

7. **Predstavitev rezultatov:** Najboljše posameznike, to je tiste z najvišjimi ocenami ustreznosti, predstavimo kot rezultat evolucijskega procesa.

Celoten algoritem je po točkah v obliki psevdokode prikazan v kodi 1.

```
BEGIN
    ustvarimo začetno populacijo
    izračun uspešnosti osebkov v populaciji

    WHILE nimamo najboljše rešitve                      OR
          nismo izvedli določenega števila korakov      OR
          ni minil določen čas                          OR
          se potomci bistveno razlikujejo od staršev
    DO
    BEGIN
        // faza selekcije
        izbor staršev za razmnoževanje

        // faza razmnoževanja
        križanje staršev
        mutiranje posameznikov

        ocena uspešnosti populacije
    END
END
```

Koda 1: Primer enostavnega genetskega algoritma

Poglavje 4

Metoda za iskanje ujemanj s pomočjo genetskih algoritmov

Osnova te naloge je metoda za iskanje preslikav, ki je predstavljena v [1]. Metodo smo si izbrali zato, ker se aktivno ukvarja s problematiko iskanja kompleksnih preslikav med shemami, ki je v današnjem času še vedno zelo aktualen problem. Poleg tega kaže velik potencial, predvsem pri združevanju z drugimi pristopi. Metoda deluje na principu evolucije – bolj natančno po principu GA. V prejšnjem poglavju smo si ogledali osnovne koncepte GA, zato da bomo v nadaljevanju lažje predstavili, kako se koncepte GA aplicira na problem iskanja ujemanj med shemami.

4.1 Opis delovanja metode

Potem ko smo razjasnili, na kakšnem principu delujejo metode GA, lahko nadaljujemo pregled delovanja metode za iskanje ujemanj med shemami z uporabo evolucijskega pristopa. Najprej bomo predstavili modeliranje problema na način, da ga je možno reševati z uporabo evolucijskih tehnik. Nato pa si bomo ogledali še implementacijo samega genetskega algoritma.

4.1.1 Modeliranje problema

Metoda omogoča iskanje vseh vrst preslikav – tako enostavnih kot kompleksnih. Ker iskanje kompleksnih preslikav še vedno ni povsem rešen problem, več pozornosti vseeno namenja iskanju kompleksnih preslikav – to je preslikav številnosti N:M, kjer se skupina elementov iz prve sheme preslika v skupino elementov druge sheme. Da bi to lahko ponazorili z drevesno strukturo, avtorji metode predlagajo združevanje elementov sheme v *izpeljane elemente sheme*.

Izpeljani elementi sheme. Naj bo $\mathcal{E}$ množica elementov podane sheme S. Izpeljani element sheme S je par $d = \langle \mathcal{F}, \pi \rangle$, kjer je $\mathcal{F} = \{f_1, \ldots, f_n\}, n > 0$ podmnožica množice $\mathcal{E}$, π pa je binarno drevo izpeljav, v katerem so listi elementi množice $\mathcal{F}$, notranja vozlišča, če obstajajo, pa so binarni ali unarni operatorji. Elementi sheme iz množice $\mathcal{F}$ se imenujejo osnovni elementi. Če so vsi osnovni elementi enakega podatkovnega tipa, potem je d *omejen izpeljani element*, ki ga definiramo z $d = \langle \mathcal{F}, \pi, \tau \rangle$, kjer τ predstavlja podatkovni tip osnovnih elementov.

Vrednost izpeljanega elementa sheme. Naj bo $d = \langle \mathcal{F}, \pi, \tau \rangle$ omejen izpeljani element sheme S, kot smo definirali zgoraj. Naj bo I_S instanca sheme S in $e \in I_S$ entiteta znotraj instance I_S. Iz tega sledi, da je $v(e, d)$ vrednost v domeni podatkovnega tipa τ, ki jo dobimo, če apliciramo zaporedje operacij, definiranih v π, na vrednosti osnovnih elementov množice $\mathcal{F}$, ki jih dobimo iz e.

Zaradi težav, ki nastopijo s primerjanjem vrednosti različnih podatkovnih tipov, bomo od tega trenutka dalje privzeli, da so vsi izpeljani elementi, s katerimi imamo opravka, omejeni. Primeri podatkovnih tipov in operacij, ki se pojavljajo v takšnih elementih, so prikazani v tabeli 4.1.

Sedaj bomo definirali *preslikavo*. Preslikava med shemama A in B je trojka $m = \langle d_A, d_B, \alpha \rangle$, kjer sta d_A in d_B izpeljana elementa istega tipa τ iz shem A in B, α pa je funkcija za ocenjevanje podobnosti, katero se lahko aplicira na podatkovne tipe τ. Funkcija α se uporablja za ocenjevanje podobnosti med d_A in d_B na entitetah, ki se pojavljajo v instancah shem A in B.

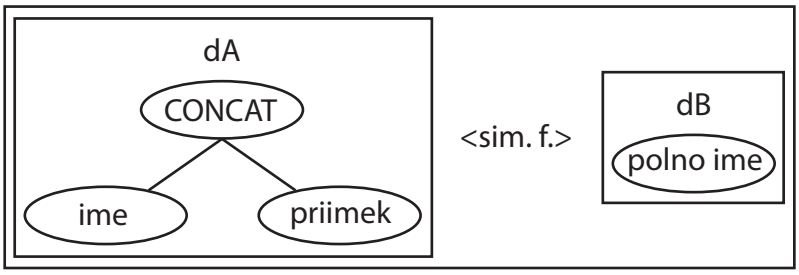
Tip podatka	Operacija	Primer
Tekst	združevanje	`CONCAT(a, b) = ab`
	ločevanje	`SPLIT(a b) = {a, b}`
Datum	seštevanje	`ADD(1.1.2015, YEAR, 2) = 1.1.2017`
	odštevanje	`SUB(1.1.2015, MONTH, 3) = 1.10.2014`
	pretvorbe	`DAYSIN(januar) = 31`
Število	seštevanje	`ADD(1, 2) = 3`
	odštevanje	`SUB(1, 2) = -1`
	množenje	`MUL(2, 3) = 6`
	deljenje	`DIV(4, 2) = 2`
	potenciranje	`EXP(2, 2) = 4`

Tabela 4.1: Nabor možnih operacij nad elementi sheme.

Obstaja mnogo funkcij za ocenjevanje podobnosti, katere lahko uporabimo pri problemu iskanja ujemanj med shemami. Za vse pa načeloma velja, da vračajo oceno podobnosti v intervalu $[0, 1]$. Ocena blizu 1 predstavlja visoko podobnost med entitetami, ocena blizu 0 pa predstavlja nizko podobnost. V primeru, da to ne drži, je treba ustrezno prilagoditi implementacijo primerjanja ocen ali pa ustrezno transformirati zalogo vrednosti funkcije v interval $[0, 1]$.

Na sliki 4.1 vidimo preprost primer preslikave med dvema shemama. Pravokotnika dA in dB predstavljata izpeljane elemente d_A in d_B iz shem A in B. Izpeljani element d_A je sestavljen iz dveh listov (osnovnih elementov) in enega vozlišča, ki predstavlja operacijo združevanja nad obema elementoma. Izpeljani element d_B je sestavljen samo iz enega elementa in je brez vozlišč.

Pri iskanju ujemanj med dvema shemama lahko za rezultat dobimo več možnih preslikav. Če pogledamo sliko 4.2, vidimo, da lahko elemente *hišna številka*, *naslov1*, *naslov2* iz sheme A preslikamo v element *naslov* iz sheme B. Prav tako lahko preslikamo *predel* iz sheme A v *območje* iz sheme B. Skupek vseh možnih preslikav ($\{m_1, \ldots, m_n\}, n > 0$) sestavlja enega posameznika, ki



Slika 4.1: Primer enostavne preslikave med shemama.

ime	priimek	hišna številka	naslov1	naslov2	predel
jane	watson	26	coranderrk street	rowethorpe	hill end
benjamin	franklin	18	derrington crescent	homewood	kingsthorpe
peter	shore	23	prescott street		bonbeach

polno ime	starost	naslov	območje
watson jen	43	coranderk 26, rowethorpe	hi end
ben franklin	33	derrington crescent 18, homewood	kingsthorpe
pete sure	39	prescott str	bonbeach

Slika 4.2: Primer instanc dveh shem in preslikav med njima.

predstavlja *možno rešitev* problema iskanja ujemanj.

Avtorji članka [1] predlagajo, da so rešitve problema posamezniki, ki se razvijajo skozi generacije od začetne populacije naključnih rešitev do končne populacije, ki predstavlja končno rešitev problema.

4.1.2 Prilagoditve evolucijskega algoritma

Predlagani algoritem deluje po principu, opisanem v poglavju 3.2. Ker je GA po zasnovi generičen, bomo vsak korak posebej umestili v ustrezen kontekst.

1. **Kreiranje začetne populacije**

V tem koraku naključno generiramo začetno populacijo osebkov. To po-

meni, da elemente obeh shem naključno sestavimo v izpeljane elemente shem. Izpeljane elemente sheme paroma združimo v preslikavo, ki ji dodelimo še funkcijo, s katero bomo ocenjevali ustreznost preslikave. Dobljene preslikave še dodatno združimo v skupine. Vsaka taka skupina, ki jo imenujemo posameznik populacije, predstavlja začetno stanje za rešitev problema, ki ga obravnavamo. Pri generiranju naključnih posameznikov je treba paziti, da generiramo takšne posameznike, ki imajo čim večjo verjetnost, da postanejo končne rešitve problema. Da to dosežemo, pa se moramo držati nekaterih omejitev, ki jih bomo predstavili v poglavju 4.1.3.

2. **Ocenjevanje posameznikov**

   Prvo in tudi vse naslednje generacije ocenjujemo s funkcijo za ocenjevanje uspešnosti. Za izbrano rešitev v trenutni populaciji $\mathcal{S} = \{m_1, m_2, \ldots, m_n\}$, kjer m_i predstavlja eno preslikavo, izračunamo njeno povprečno uspešnost na naslednji način:

$$f(\mathcal{S}) = \frac{\sum_{i=1}^{n} eval(m_i)}{n} \tag{4.1}$$

   V enačbi 4.1 je m_i preslikava, ki jo zapišemo kot $m_i = \langle a_i, b_i, \alpha_i \rangle$, kjer sta a_i in b_i izpeljana elementa shem A in B, α_i pa je funkcija za ocenjevanje podobnosti. Naloga funkcije *eval* je izračun ocene podobnosti s pomočjo izbrane metrike med izpeljanima elementoma a_i in b_i glede na podatke v instancah shem A in B. To v praksi pomeni, da se liste (osnovne elemente) v izpeljanih elementih zamenja z ustreznimi podatki iz instanc, nato pa se te podatke združi glede na definirane operacije v vozliščih drevesa. Nad dobljenima rezultatoma se nato izvede funkcija za ocenjevanje podobnosti, ki nam da oceno uspešnosti trenutne preslikave.

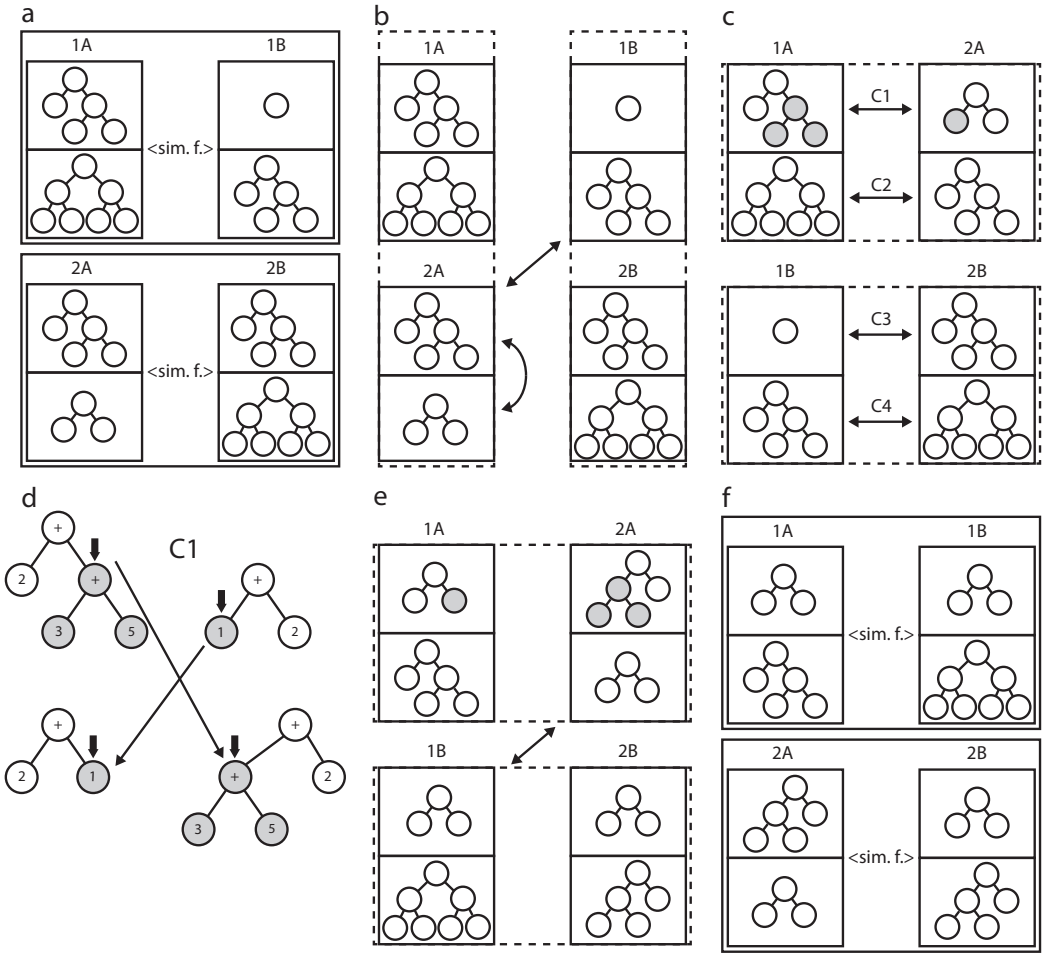
3. **Preverjanje ustavitvenega pogoja**

   Postopek evolucije se izvaja toliko časa, dokler ne dosežemo določenega števila iteracij, ki ga definiramo s parametrom k. Ko je ta pogoj dosežen,

se ne glede na doseženo oceno posameznikov v trenutni populaciji izvajanje algoritma prekine. V primeru, da populacija doseže postavljen prag, se bo algoritem ustavil, še preden bo izpolnjen glavni ustavitveni pogoj.

4. **Evolucija populacije**

Kot smo že omenili, novo generacijo v evolucijskem procesu dobimo z upoštevanjem izračunane ocene uspešnosti za vsakega posameznika v populaciji. Na osnovi te ocene se v našem primeru v enem koraku izvede korake 4, 5, in 6 iz algoritma, opisanega v poglavju 4. Proces evolucije populacije se začne tako, da se iz trenutne populacije izbere N posameznikov z najvišjimi ocenami in se jih prenese v novo generacijo. Nato se izbere $M < N$ posameznikov, ki predstavljajo starše posameznikov, ki bodo sestavljali preostali del nove generacije. Izbrani posamezniki so podvrženi operacijama *križanja* in *mutacije*. Obe operaciji se izvaja samo na izbranih posameznikih. Operacija križanja omogoča izmenjavo komponent med izbranima posameznikoma, operacija mutacije pa deluje tako, da naključno spreminja strukturo izpeljanih elementov shem. To stori tako, da v strukturo posameznika na naključno izbrano mesto vstavi naključno generirano drevo. Najprej se izvede križanje in šele nato se izvede mutiranje novih, s križanjem dobljenih posameznikov. Čeprav se te operacije izvajajo naključno, pri tem vseeno pazimo, da se tip izpeljanega elementa ne spreminja (omejeni izpeljani elementi). Na sliki 4.3 je prikazan postopek, kako operacija križanja ustvarja nove posameznike v izbrani metodi. Kot je razvidno iz same slike, se najprej izbereta dva posameznika, nad katerima se bo izvajala celotna operacija (a). Nato sledi kreiranje parov delov posameznikov, med katerimi se bo prenašal genetski material (b). To pomeni, da se dele posameznikov, ki spadajo v isto shemo, združi skupaj. V naslednjem koraku se izvede križanje (c in d). Ko se križanje zaključi, se ustvarijo novi posamezniki, tako da se ustvarijo nove preslikave med njimi (e in f). To naredimo tako, da se naključno poveže dele posameznikov, ki spadajo v eno shemo, z deli posameznikov, ki spadajo v drugo shemo.



Slika 4.3: Postopek križanja.

5. **Predstavitev rezultatov**

V tem koraku ob koncu izvajanja algoritma izmed vseh posameznikov izberemo tistega z najvišjo oceno ustreznosti in ga s pomočjo ustrezne vizualizacije predstavimo uporabniku v obliki preslikav med drevesi elementov prve in druge sheme.

4.1.3 Omejitve evolucijskega procesa

Evolucijski algoritem, predstavljen v poglavju 3.2, je zelo splošen in fleksibilen, kar mu omogoča iskanje rešitev v velikem prostoru. Vendar pa je pri problemu iskanja ujemanj med shemami vseeno treba paziti na velikost preiskovalnega prostora, saj želimo rešitve dobiti v realnem času. Zato je v ta proces vpeljanih nekaj omejitev, ki poskrbijo, da je preiskovalni prostor čim manjši.

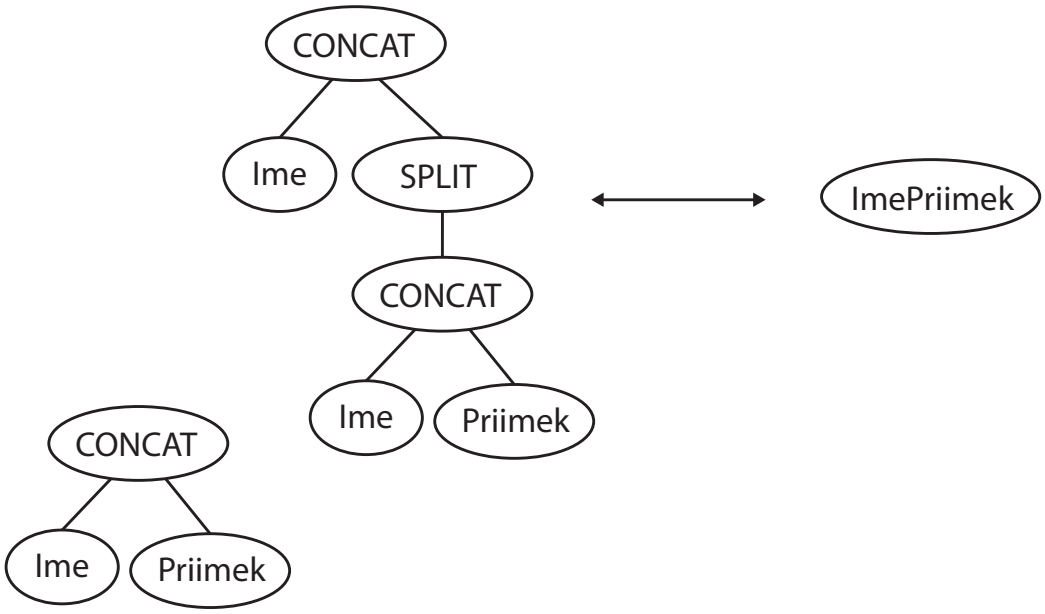
Omejitev tipa podatkov

Prva in glavna omejitev, o kateri smo že spregovorili, je omejevanje izpeljanih elementov sheme na izbrani tip podatkov. To ne samo omeji število elementov sheme, izmed katerih izbiramo kandidate za tvorbo preslikav, ampak omeji tudi število operacij, ki lahko zasedajo notranja vozlišča drevesne strukture.

Preprečevanje podvajanja elementov

Pri kreiranju začetne populacije moramo preprečiti podvajanje elementov v drevesih. Pomembno je, da se ta omejitev upošteva tudi pri izvajanju vseh nadaljnjih operacij križanja in mutacije, sicer v nasprotnem primeru kot rešitev dobimo povsem nesmiselne preslikave. S pojmom nesmiselne preslikave mislimo na tiste preslikave, ki so sicer povsem veljavne, vendar vsebujejo veliko več elementov in operacij, kot je potrebno. Primer takšne preslikave je prikazan na sliki 4.4. Preslikava na sliki je veljavna, vendar bi bilo dovolj, da bi bilo na levi strani drevo sestavljeno samo iz dveh elementov *Ime* in *Priimek* in ene operacije `CONCAT`.

Pri generiranju in mutiranju dreves se ni težko držati te omejitve. Tam samo poskrbimo, da ko enkrat element uporabimo, ga izločimo iz nabora možnih izbir. Težje je pri izvajanju operacije križanja, saj tu prenašamo naključno izbrana poddrevesa med dvema drevesoma, saj v primeru dvojnikov ne vemo, katerega zavreči. V naši implementaciji smo to rešili tako, da izmed podvojenih elementov odstranimo naključno izbranega, tako da ga



Slika 4.4: Primer nesmiselne preslikave.

nadomestimo s praznim vozliščem. V izogib velikemu številu praznih vozlišč v drevesu nato izvedemo še operacijo poenostavljanja dreves, ki prazna vozlišča odstrani.

Omejitev višine dreves

Tretja in zadnja omejitev je bolj enostavna, a ima pri iskanju preslikav velik pomen na samo hitrost izvajanja algoritma. To je omejitev višine drevesne strukture. Z višino drevesa označujemo število povezav od korena drevesa do najnižjega lista. Drevo s samo enim vozliščem ima višino 0. Razlog za vpeljavo te omejitve temelji na predpostavki, da je odvisno od problema, kakšne največje številnosti preslikav je možno pričakovati kot rešitve. V določenih primerih obstajajo preslikave številnosti N:M, kjer sta N in M zelo velika, v nekaterih drugih primerih sta N in M majhna (npr. največ 10). Zato, če npr. višino drevesa omejimo na 3, lahko kot rezultat pričakujemo preslikave številnosti največ 8:8. V praksi se izkaže, da je to povsem dovolj. Če to omejitev izpustimo, se lahko zgodi, da bo algoritem zabredel v preiskovanje

zelo kompleksnih preslikav, v katerih v večini primerov ne najde ustreznih rešitev. Poleg tega se s tem izredno poveča čas izvajanja, saj časovna kompleksnost operacij narašča z logaritmom drevesa ($\mathcal{O}(\log n)$), v najslabšem primeru $\mathcal{O}(n)$.

4.1.4 Strategije za ocenjevanje podobnosti

Kot je v navadi pri vseh evolucijskih metodah, je tudi naša izbrana metoda zelo odvisna od strategije za ocenjevanje podobnosti med posamezniki. V našem primeru naj bo $\mathcal{S} = \{m_1, m_2, \ldots, m_n\}$ predlagana rešitev problema, ki jo želimo oceniti, kjer je $m_i = \langle a_i, b_i, \alpha_i \rangle$ preslikava sestavljena iz izpeljanih elementov a_i sheme A in b_i sheme B ter funkcije za ocenjevanje podobnosti α_i. Funkcija, predstavljena v enačbi 4.1, izračuna povprečno oceno ustreznosti vseh preslikav predlagane rešitve z uporabo funkcij α_i za ocenjevanje podobnosti. Zdaj je vprašanje, kako uporabiti podatke, ki so nam na voljo, za ocenjevanje podobnosti. Avtorji metode navajajo dva pristopa: entitetno orientirano strategijo, ki temelji na metodi razdvojevanja podatkov, predstavljeni v [2], in vrednostno orientirano strategijo, ki je osnovana na tehniki pridobivanja informacij.

Entitetno orientirana strategija je osnovana na naslednji predpostavki: če obstajajo skupne entitete med instancami shem, med katerimi iščemo ujemanja, potem bomo v primeru najdenih pravilnih preslikav take instance tudi prepoznali. Naj I_A in I_B predstavljata instanci shem A in B, med katerima iščemo ujemanja. Poleg tega naj obstajata tudi množica $\{e_1, \ldots, e_n\}$ iz I_A in množica $\{f_1, \ldots, f_n\}$ iz I_B, za kateri velja, da je $e_j = f_j (1 \leq j \leq n)$. Potemtakem mora obstajati ena ali več preslikav $m = \langle d_A, d_B, \alpha \rangle$, za katere velja $\alpha(v(d_A, e_j), v(d_B, f_j)) \cong 1$, kjer sta $v(d_A, e_j)$ in $v(d_B, f_j)$ vrednosti izpeljanih elementov d_A in d_B v entitetah e_j in f_j. Kot primer vzemimo preslikavo, prikazano na sliki 4.1, ki se navezuje na shemi iz slike 4.2. Naj bo t_a terka iz tabele A in t_b terka iz tabele B, tako da obe terki predstavljata isto entiteto. Če združimo vrednosti *imena* in *priimka* iz terke t_a in dobljeno primerjamo s *polnim imenom* iz terke t_b, lahko pričakujemo da bo ocena podobnosti pri

uporabi metrike Jaro-Winkler blizu 1. Metriko Jaro-Winkler na tem mestu omenjamo zato, ker so avtorji izbrane metode iz [1] to metriko največ uporabljali pri testiranju metode. Zato si na kratko poglejmo, kako deluje.

Metrika *Jaro-Winkler*[22] je dobro poznana funkcija za ocenjevanje podobnosti med kratkimi besedami (predvsem imeni in priimki). Metrika Jaro-Winkler je razširjena oblika metrike Jaro [24]. Metrika Jaro podobnost med besedami ocenjuje na osnovi števila skupnih črk in zaporedja, v katerem si sledijo [23]. Definirajmo niza $s = a_1, \ldots, a_K$ in $t = b_1, \ldots, b_L$. Če je črka a_i iz a prisotna v t, potem obstaja $b_j = a_i$ v t, tako da velja $i - H \leq j \leq i + H$, kjer je $H = \frac{\min(|s|,|t|)}{2}$. Naj bo $s' = a'_1, \ldots, a'_K$ množica črk iz s, ki so v t (v takem zaporedju, kot se pojavljajo v s), in naj bo $t' = b'_1, \ldots, b'_L$ množica črk iz t, ki so v s. Sedaj definiramo transpozicije za s' in t' kot pozicijo i, tako da velja $a'_i \neq b'_i$. Naj bo $T_{s',t'}$ polovično število vseh transpozicij za s' in t'. Metriko Jaro za niza s in t nato definiramo kot

$$Jaro(s,t) = \frac{1}{3} \cdot \left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{|s'|} \right) \tag{4.2}$$

Metrika Jaro-Winkler izboljša zgornjo enačbo tako, da vpelje dodatno "nagrado", ki je odvisna od dolžine skupne predpone P nizov s in t. Če je dolžina skupne predpone obeh nizov daljša, bo ocena podobnosti večja. Naj bo $P' = \max(P, 4)$. Iz tega sledi, da je enačba za metriko Jaro-Winkler sledeča:

$$Jaro\text{-}Winkler(s,t) = Jaro(s,t) + \frac{P'}{10} \cdot (1 - Jaro(s,t)) \tag{4.3}$$

Iz vsega tega sledi, da je entitetno orientirana strategija močno odvisna od medsebojne podobnosti podatkovnih instanc.

Za razliko od entitetno orientirane strategije, ki ujemanja išče samo na podlagi podatkov, ki so skupni obema instancama, *vrednostno orientirana strategija* deluje tako, da primerja množico vrednosti izpeljanih elementov iz obeh instanc podatkov, ne glede na njihovo podobnost. Zato je ta strategija bolj primerna za primere, kadar je presek med instancami zanemarljiv ali pa sploh ne obstaja. Naj bosta d_A in d_B izpeljana elementa shem A in B,

ki sta oba istega podatkovnega tipa τ. Poleg tega imamo tudi dve instanci I_A (za shemo A) in I_B (za shemo B). Naj za shemo $\mathcal{X} \in \{A, B\}$ velja, da je $\mathcal{V}(d_X) = \{v(d_X, e) \mid e \in I_X\}$ množica vseh vrednosti d_X, pridobljenih iz entitet instance I_X, kjer je $v(d_X, e)$ vrednost $d_\mathcal{X}$ v entiteti e. Če sta I_A in I_B reprezentativni instanci, potem lahko z veliko verjetnostjo trdimo, da obstaja ena ali več preslikav $m = \langle d_A, d_B, \alpha \rangle$, za katere velja, da sta $\mathcal{V}(d_A)$ in $\mathcal{V}(d_B)$ podobni glede na primerno funkcijo za ocenjevanje podobnosti α.

Za primer se spet obrnimo na vzorčno preslikavo iz slike 4.1 nad podatki iz slike 4.2. Z uporabo zgoraj definirane notacije dobimo:

- $\mathcal{V}(d_A) =$ "Jane Watson", "Benjamin Franklin", "Peter Shore"
- $\mathcal{V}(d_B) =$ "Watson Jen", "Ben Franklin", "Pete Sure"

Iz primera vidimo, da si posamezni elementi niso 100 % podobni. Kljub temu lahko z ustrezno izbrano metriko za rezultat dobimo visoko oceno ujemanja. Za razliko od entitetno orientirane stretegije, ki za delovanje v obeh instancah potrebuje isti (ali zelo podoben) zapis, za to strategijo zadošča že, da instanci obeh shem vsebujeta le nekaj skupnih besed.

Opazimo, da metrika Jaro-Winkler za ocenjevanje podobnosti v tem primeru ni primerna. Ta metrika je skupaj z Levenshteinovo razdaljo [25] bolj primerna za ocenjevanje podobnosti med posameznimi nizi. Za ocenjevanje podobnosti med množicami, kot imamo v primeru vrednostno orietirane strategije, bolj pride v poštev Jaccardova razdalja (angl. *Jaccard distance*) ali metrika TF-IDF (angl. *term frequency-inverse document frequency*).

S spremembo metrike za ocenjevanje podobnosti se spremeni tudi sam način izvajanja ocenjevanja podobnosti in ustreznosti preslikave [1]. Z uporabo vrednostno orientirane metrike za ocenjevanje podobnosti se funkcija *eval* iz enačbe 4.1 spremeni v

$$eval(m) = \alpha\left(\mathcal{V}\left(d_A\right), \mathcal{V}\left(d_B\right)\right) \tag{4.4}$$

kjer sta $\mathcal{V}(d_A)$ in $\mathcal{V}(d_B)$ množici vrednosti izpeljanih elementov, ki smo jih definirali v prejšnjih odstavkih. Da bo bolje razumljivo, kako uporabiti vre-

dnostno orientirano metriko, bomo to predstavili na primeru metrike TF-IDF, ki jo uporabljamo tudi v naši metodi. Da bodo matematične definicije bolj razumljive na pogled, bomo izraza za množici $\mathcal{V}(d_A, e_j)$ in $\mathcal{V}(d_B, f_j)$ zamenjali z $\mathcal{V}_A$ in $\mathcal{V}_B$. Prav tako bomo delali s predpostavko, da so elementi v obeh množicah tekstovnega tipa.

Naj bo $\mathcal{T}_X$ množica vseh besed, ki se pojavljajo v $\mathcal{V}_X (X \in \mathcal{A}, \mathcal{B})$, in naj bo $\mathcal{T} = \mathcal{T}_A \cup \mathcal{T}_B$. Podobnost TF-IDF med $\mathcal{V}_A$ in $\mathcal{V}_B$ se potem izračuna tako:

$$sim(\mathcal{V}_A, \mathcal{V}_B) = \frac{\sum_{t \in \mathcal{T}} w(t, \mathcal{V}_A) \cdot w(t, \mathcal{V}_B)}{\sqrt{\sum_{t \in \mathcal{T}} w(t, \mathcal{V}_A)^2 \cdot \sum_{t \in \mathcal{T}} w(t, \mathcal{V}_B)^2}} \tag{4.5}$$

kjer $w(t, \mathcal{V}_X)$ predstavlja izraz

$$w(t, \mathcal{V}_X) = f(t, \mathcal{V}_X) \cdot idf(t) \tag{4.6}$$

kjer je $f(t, \mathcal{V}_X)$ normalizirana frekvenca pojavitve besede t v $\mathcal{V}_X$, $idf(t)$ pa je vrednost, znana kot inverzna frekvenca besede t v dokumentu (angl. *inverse document frequency*), ki prikazuje pomembnost besede t v množicah $\mathcal{V}_A$ in $\mathcal{V}_B$. Vrednost $f(t, \mathcal{V}_X)$ je podana z enačbo

$$f(t, \mathcal{V}_X) = \frac{freq(t, \mathcal{V}_X)}{\max_u \{freq(u, \mathcal{V}_X)\}} \tag{4.7}$$

kjer je $freq(t, \mathcal{V}_X)$ število pojavitev besede t v $\mathcal{V}_X$, $\max_u \{freq(u, \mathcal{V}_X\}$ pa je največja frekvenca pojavitev med vsemi besedami v $\mathcal{V}_X$. Kot zadnja pa je vrednost $idf(t)$ podana kot

$$idf(t) = \log \frac{N}{n(t)} \tag{4.8}$$

kjer je N število vseh elementov sheme, ki jih pregledujemo, $n(i)$ pa je število izpeljanih elementov sheme d, pri katerih se beseda t pojavlja v $\mathcal{V}(d)$.

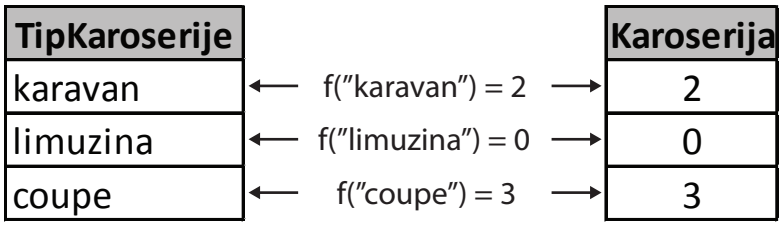
Poglavje 5

Izboljšana metoda

5.1 Identifikacija možnih izboljšav

Izbrana metoda iz [1] deluje na osnovi instanc podatkov in predpostavlja, da so instance v neki meri podobne – da vsebujejo vsaj nekaj zapisov, ki predstavljajo iste entitete. Kaj pa če ta predpostavka ne vzdrži in imamo na voljo samo instance s povsem različnimi podatki? Kaj če podatkovne instance sploh ne obstajajo? V takih primerih nam bo metoda vračala nezanesljive preslikave ali pa jih sploh ne bo uspela generirati. Ker obstajajo še drugi načini iskanja ujemanj med shemami, je možnost, da slabost izbrane metode nekoliko omilimo. Kadar pride do scenarija, da iz podatkov ni moč ugotoviti preslikav, lahko posežemo po informacijah, ki se skrivajo v sami shemi. Zato predlagamo izboljšavo metode na način, da se jo ustrezno preoblikuje v hibridno metodo, ki išče ujemanja na podlagi podatkovnih instanc ter podatkov sheme.

Do sedaj smo večinoma govorili samo o pristopih, ki nam omogočajo iskanje preslikav med tekstovnimi podatki. Razen par omenjenih primerov dela s številskimi podatki v literaturi ni nikjer navedeno, na kakšen način bi k temu zares pristopili. Pojavi se problem, saj opisanih pristopov ni enostavno prenesti na druge tipe podatkov. Zato predlagamo, da se izbrana metoda izboljša tako, da se doda možnost iskanja ujemanj tudi med drugimi tipi



Slika 5.1: Primer kategoriziranih podatkov.

podatkov, predvsem številskimi.

Še ena možna izboljšava metode v smislu iskanja ujemanj med drugimi tipi podatkov je obravnavanje kategoriziranih podatkov (slika 5.1). To so podatki, ki v nekem elementu lahko zasedajo eno od določenih vrednosti, ki so lahko v eni shemi predstavljene opisno (npr. limuzina, kombilimuzina, terenec ...), v drugi pa na primer s šiframi (npr. 1, 2, 3 ...). V večini primerov take podatke srečamo v šifrantih [27].

5.2 Izboljšave

5.2.1 Uporaba podatkov sheme

Preden smo lahko začeli implementacijo, smo se morali odločiti, katere podatke sheme bomo uporabili. Glede na to, da izbrana metoda že deloma deluje po principu omejitev, saj omejuje iskanje preslikav samo nad podatki izbranega tipa, smo se odločili, da za izboljšavo iskanja ujemanj uporabimo imena elementov. Da bi lahko določili, kateri elementi so si po nazivu sodeč najbolj podobni, smo najprej določili, katere funkcije za ocenjevanje podobnosti bomo uporabili [12]. Uporabili smo naslednje funkcije:

- **Soundex**: algoritem, ki izračuna fonetično podobnost med besedami na podlagi njihovega soundex kodiranja. Uporabno predvsem za preverjanje podobnosti med besedami, ki se drugače zapišejo in podobno izgovorijo (npr. Soundex(“butale”) = B340 ≅ Soundex(“budale”) = B340)

- **Metaphone**: deluje na podoben način kot Soundex, le da je bolj natančen [9]

- **Levenshtein**: algoritem, ki meri število razlik med dvema besedama (minimalno število operacij, da se prva beseda pretvori v drugo)

- **N-gram**: podobnost med besedami glede na število skupnih podnizov (npr. 2-gram besede "pesem" je {pe, es, se, em}, 3-gram pa {pes, ese, sem}).

Ker uporabljene funkcije ne vračajo vrednosti na intervalu $[0, 1]$, moramo dobljene vrednosti najprej normalizirati. Za vsak element prve sheme izračunamo podobnost z vsakim elementom druge sheme. To storimo tako, da na podlagi vseh zgoraj navedenih funkcij izračunamo podobnost, kot rezultat pa vrnemo povprečje dobljenih podobnosti. Dobljene ocene podobnosti nato uporabimo pri generiranju naključnih rešitev v prvi in vseh naslednjih populacijah. Drevesa se tako kot prej še vedno generirajo naključno, vendar imajo pri generiranju večjo verjetnost izbire tisti elementi, ki imajo večjo povprečno podobnost z elementi druge sheme (angl. *weighted random sampling*). S tem dosežemo, da že v prvi populaciji dobimo čim bolj verjetne preslikave.

Ker se je po začetnem testiranju izkazalo, da samo omejevanje izbire elementov glede na podobnost njihovih imen ni dovolj za izboljšanje rezultatov, smo izračunane ocene podobnosti uporabili še drugje. In sicer smo se odločili, da bomo dobljene informacije uporabili pri samem postopku ocenjevanja uspešnosti posameznih osebkov in ne samo pri generiranju možnih preslikav. Tako pri entitetno kot vrednostno orientiranem pristopu smo dodali dodaten korak, kjer izračunamo uspešnost preslikav na podlagi podobnosti med nazivi elementov, ki sestavljajo preslikavo. To storimo tako, da za elemente dreves prve in druge sheme v vsaki preslikavi izračunamo medsebojno povprečno oceno podobnosti (koda 2). Oceno uspešnosti preslikave sedaj izračunamo po enačbi

$$\begin{aligned} fitness = dataSim \cdot dataWeight + attrSim \cdot attrWeight \\ dataWeight + attrWeight = 1 \end{aligned} \tag{5.1}$$

kjer *dataSim* predstavlja oceno podobnosti, ki jo pridobimo z analizo podatkovnih instanc, *attrSim* predstavlja oceno podobnosti med elementi shem, *dataWeight* in *attrWeight* pa sta uteži, ki določata, katera ocena podobnosti ima večji vpliv. Uteži po lastni presoji določi uporabnik. Če so elementi obeh shem že na prvi pogled zelo različni, je potem bolje, da uporabnik utež *attrWeight* nastavi na nižjo vrednost kot utež *dataWeight*. Privzeto sta vrednosti obeh uteži nastavljeni na 0.5. Z vpeljavo dodatnega računanja podobnosti med nazivi elementov sheme smo razširili obstoječo metodo v hibridno metodo.

5.2.2 Podpora za dodatne podatkovne tipe

Izbrana metoda v splošnem podpira samo iskanje preslikav med podatki tekstovnega tipa. Posredno je možno tudi iskanje ujemanj med numeričnimi tipi podatkov, vendar se iskanje izvaja na osnovi funkcij za ocenjevanje podobnosti, ki pa so v praksi namenjene samo ocenjevanju podobnosti med nizi. Zato je pričakovano, da metoda takih preslikav v večini primerov ni sposobna najti ali pa najde zelo nezanesljive preslikave.

Reševanja tega problema smo se lotili postopoma. Začeli smo z entitetno orientiranim pristopom pri katerem smo vpeljali novo funkcijo za ocenjevanje podobnosti – to je Evklidska razdalja.

Pri vrednostno orientiranemu pristopu smo k problemu pristopili podobno kot avtorji originalne metode, ki so predlagali TF-IDF funkcijo za ocenjevanje podobnosti med množicami. Ker TF-IDF za števila ni primeren, smo najprej poskusili z Jaccardovo metriko. Ta se je izkazala za primerno. Ker smo želeli še bolj izboljšati rezultate, smo se odločili, da poskusimo še nekaj bolj kompleksnega. V članku [10] so avtorji predlagali način, kako prilagoditi TF-IDF, da deluje tudi za številske podatke. Naj bo $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$

```
elListS1; // elementi drevesa sheme S1 v preslikavi M
elListS2; // elementi drevesa sheme S2 v preslikavi M

avgSim = 0
for elS1 in elListS1
{
        for elS1 in elListS2
        {
                // izračun podobnosti med elS1 in elS2
                // SOUNDEX + (Double)Metaphone + Levenshtein +
                // + Di-Gram
                avgSim += sim(elS1, elS2)
        }

        // izračun povprečne podobnosti med elS1 in elListS2
        avgSim += avg(avgSim)
}

// izračun skupne povrečne podobnosti med elListS1 in elListS2
avgSim = avg(avgSim)
```

Koda 2: Psevdokoda za izračun podobnosti imen elementov

množica vrednosti, ki jih dobimo z evalvacijo izpeljanih elementov sheme. Za vsak element množice $\mathcal{T}$ definiramo funkcijo

$$idf(t) = \log \left(\frac{n}{\sum_{i}^{n} e^{-\frac{1}{2}\left(\frac{t_i - t}{h}\right)^2}} \right) \qquad t, t_i \in \mathcal{T} \tag{5.2}$$

kjer n predstavlja število vseh elementov množice $\mathcal{T}$, h pa je parameter, ki ga je treba pametno določiti. Imenovalec predstavlja frekvenco t_i, to je vsota "prispevkov" številu t od vsakega števila t_i. Prispevki so modelirani kot skalirane Gaussove porazdelitve, tako da bolj ko je t oddaljen od t_i, manjši je prispevek števila t_i številu t. Zgornja izpeljava formule je bila pridobljena z neparametričnim ocenjevanjem funkcij gostote verjetnosti s pomočjo KDE. Avtorji navajajo, da je primerna vrednost za parameter h $1.06 \cdot \sigma \cdot n^{-\frac{1}{5}}$, kjer je σ standarni odklon množice $\mathcal{T}$.

Ko smo to funkcijo uspešno aplicirali, smo v testne namene poskusili prilagoditi še obstoječo funkcijo TF-IDF, da bi delovala s števili. Vse, kar smo spremenili je, da sedaj števila obravnavamo kot besede.

Za konec smo implementirali Jaccardovo metriko za ocenjevanje podobnosti. Tudi uporaba te funkcije daje dobre rezultate, kar pa je najbolj važno, je, da je dosti hitrejša kot TF-IDF.

5.3 Implementacija

Izbrano metodo smo implementirali v programskem jeziku Java. Iskanje ujemanj izvajamo na shemah relacijskih baz namesto na večkrat omenjenih shemah XML. Sam izvor podatkov ne igra nobene vloge, saj se tako podatki iz relacijskih shem kot iz shem XML transformirajo v obliko, ki je primerna za naš pristop. Za testiranje smo uporabili relacijsko bazo MySQL. Samo iskanje začnemo tako, da najprej v konfiguraciji nastavimo, katere vrste preslikav

iščemo (npr. STRING, NUMBER ...), določimo omejitev števila generacij in nastavimo, koliko preslikav naj vsebuje vsak posameznik. Ko program konča izvajanje, nam kot rešitev ponudi najboljšega posameznika oziroma najboljšo množico preslikav, ki jih vsebuje omenjeni posameznik. Ker se lahko zgodi, da je število vseh možnih preslikav večje od števila preslikav, ki jih lahko vsebuje posameznik, nam program zato ne bo pokazal vseh možnih preslikav. Možno je tudi, da bo kot rešitev predlagal več istih preslikav. V tem primeru moramo v konfiguraciji ročno nastaviti, katere elemente naj se ob naslednjem zagonu izpusti, da se zmanjša množico možnih preslikav. To ponavljamo, dokler ne dobimo vseh preslikav oziroma dokler ne zmanjka elementov, med katerimi iščemo ujemanja.

Poglavje 6

Vrednotenje in primerjava predlagane metode

V tem poglavju najprej predstavimo, na kakšen način bomo izvajali teste, ki bodo služili kot primerjava med našo izboljšano metodo in originalno metodo, nato pa pokažemo rezultate teh testov in jih kometiramo.

Kadar iščemo ujemanja med shemami samo na podlagi njihovih instanc, se vedno srečamo z enim od naslednjih scenarijev:

1. Instanci sheme A in B se **delno prekrivata**

   Cilj testiranja je ugotoviti, kako se metoda obnaša, kadar instanci obeh shem hranita deloma podobne podatke (npr. 50 % podatkov si delita). Tu upoštevamo, da lahko podatki vsebujejo napake (npr. tipkarske) in da so lahko predstavljeni na drugačen način. To je primer scenarija, ki se v praksi največkrat pojavlja.

2. Instanci sheme A in B se **popolnoma prekrivata**

   Cilj testiranja je ugotoviti, kako se metoda obnaša, kadar instanci vsebujeta popolnoma enake podatke – za vsako entiteto iz instance sheme A obstaja enaka entiteta v instanci B.

3. Instanci sheme A in B se **ne prekrivata**

Cilj testiranja je ugotoviti, kako se metoda obnaša, kadar instanci vsebujeta popolnoma različne podatke. To je najtežji scenarij za vse metode, ki iščejo ujemanja samo na podlagi instanc.

Pri testiranju scenarija 1 in 2 se praviloma uporablja entitetno orientirani pristop za ocenjevanje pravilnosti preslikav. Kot smo že omenili, je razlog za to podvojevanje podatkov [31]. Če naš pristop ugotovi, da obstajajo podatki, za katere lahko z veliko verjetnostjo trdimo, da predstavljajo enake entitete, potem smo posredno našli tudi ustrezne preslikave med njimi. Entitetno orientirani pristop je v tem primeru bolj primeren in bolj natančen, kar pa še ne pomeni, da ne moremo uporabljati vrednostno orientiranega pristopa.

Ravno nasprotno pa je pri scenariju 3. Ker obstoja podvojenih podatkov ne moremo potrditi, tudi preslikav ne moremo generirati. Zato se za ta scenarij uporablja samo vrednostno orientirani pristop, ki poskuša najti semantične povezave med instancami. Se pravi, da ne išče direktnih ujemanj, ampak je dovolj, da ima množica podatkov dovolj podoben nabor elementov kot druga množica.

6.1 Eksperimentalni protokol in parametri

Avtorji originalne metode so za izvajanje testov prvotno uporabili samo dve shemi, ki sta dostopni na spletni strani ustanove Illinois Semantic Integration Archive [14]. To sta t. i. "Real State" in "Inventory" shemi. Ker njuni instanci ne vsebujeta skupnih podatkov, se ti shemi uporabljata samo za testiranje 3. scenarija. Obe instanci vsebujeta nekaj več kot 100 zapisov.

Poleg teh dveh shem so avtorji generirali tudi sintetične oziroma umetne sheme in instance. To so dosegli z uporabo generatorja podatkov SDG (*Synthetic Dataset Generator*), ki se nahaja v programskem paketu Febrl [15]. Ker v času nastajanja te naloge nismo uspeli pognati tega generatorja, smo posegli po znani spletni storitvi GenerateData [16]. S pomočjo te storitve smo generirali enake sheme, kot so jih uporabljali avtorji originalne metode. Osnovni elementi, ki sestavljajo generirane sheme, so: *forename*, *surname*,

street_number, *address1*, *address2*, *suburb*, *postal_code*, *state*, *date_of_birth*, *age*, *phone_number* in *social_security_number*. Generirali smo dve shemi, ki smo ju uporabili za testiranje vseh treh scenarijev. Za vsak scenarij smo ločeno generirali instance, in sicer na naslednji način:

1. **delno prekrivanje**: generirali smo 100 zapisov za vsako shemo z omejitvijo, da se 10 % podatkov nahaja v obeh instancah,

2. **popolno prekrivanje**: generirali smo 100 zapisov za vsako shemo, s tem da se vsi podatki nahajajo v obeh instancah,

3. **ni prekrivanja**: generirali smo 100 in 1000 zapisov za vsako shemo, tako da instanci nimata skupnih podatkov.

S samo 100 zapisi se v praksi verjetno ne bi nikoli srečali. 100 zapisov smo uporabili samo zato, ker testni podatki za tretji scenarij, ki so dostopni na [14], vsebujejo ravno toliko zapisov. Ker nismo hoteli delati razlik, smo za vse tri scenarije vzeli enako število zapisov. Za testne primere ta številka povsem zadošča, predvidevamo pa, da bi v primeru testiranja na večji množici podatkov metoda dajala boljše rezultate – večja ko je množica podatkov, večja je verjetnost, da se podatki znotraj množice podvajajo. Prav zato smo v nadaljevanju za tretji scenarij dodatno generirali še 1000 zapisov, da bi preverili, ali naša trditev drži.

Da ne bi bile vse preslikave samo enostavne (številnost 1:1), smo tako kot avtorji originalne metode izbrane elemente v shemah združili (npr. `CONCAT(`*forename*, *surname*`)`, `CONCAT(`*address1*, *address2*, *street_number*`)`.

Rezultate testiranja ovrednotimo s pristopom, ki so ga pri testiranju uporabili tudi avtorji originalne metode (in ki so ga uporabili tudi avtorji drugih člankov). S tem vidimo, kakšno natančnost dosega testirana metoda. Formula za izračun ocene natančnosti metode je naslednja:

$$natančnost = \frac{št.\ pravilno\ najdenih\ preslikav}{št.\ vseh\ preslikav} \tag{6.1}$$

	GEN:100	GEN:1000	Real State	Inventory
Št. elementov v shemi A	12	12	32	44
Št. elementov v shemi B	7	7	19	38
Št. 1:1 preslikav	7	7	7	27
Tekstovne preslikave	3	3	6	11
Numerične preslikave	4	4	1	16
Št. kompleksnih preslikav	2	2	12	11
Tekstovne preslikave	2	2	5	4
Numerične preslikave	0	0	7	7

Tabela 6.1: Statistika elementov in preslikav v uporabljenih shemah.

Pred začetkom testiranja smo preizkušali različne parametre na vseh treh scenarijih, vendar da bi dobili karseda dobro primerjavo z originalno metodo, smo pri samem testiranju nato uporabili iste parametre, kot so jih navedli avtorji originalne metode. Uporabljeni parametri so navedeni v tabeli 6.2. Testiranje smo izvedli na stacionarnem računalniku s procesorjem Intel Core i7 920 (4 jeder, 2,66 GHz), 12 GB pomnilnika in navadnim trdim diskom (7200 RPM). Vse sheme in instance smo hranili v bazi MySQL 5.6.

6.2 Izvedba testiranja in rezultati

6.2.1 Primerjava z originalno metodo

Scenarij 1: delno prekrivanje podatkov

Prvi del testiranja se izvaja na podatkih z delnim prekrivanjem. Ker se v praksi v večini primerov preslikave išče ravno med shemami, katerih instance hranijo vsaj deloma podobne podatke, lahko s precejšnjo gotovostjo trdimo,

Parameter	**1**	**2**	**3**
Max. št. generacij	100	100	100
Velikost populacije	100	100	200
Reprodukcija	20 %	20 %	20 %
Križanje	80 %	80 %	80 %
Mutiranje	2 %	2 %	2 %
Omejitev globine dreves	3	3	3
Ocenjevanje podobnosti	Levenshtein, evklidska razdalja	Sort Jaro-Winkler, evklidska razdalja	TF-IDF, Jaccard
Meja podobnosti	0.90	0.85	0.95
Velikost posameznika	6	6	6

Tabela 6.2: Vrednosti parametrov, uporabljenih pri testiranju (1 – delno prekrivanje, 2 – popolno prekrivanje, 3 – brez prekrivanja podatkov).

da so dobljeni rezultati dober pokazatelj, kako bi metoda delovala v praksi.

Za iskanje preslikav smo v tem primeru uporabili entitetno orientirani pristop, ker si instanci shem delita podatke. Za ocenjevanje podobnosti med tekstovnimi podatki smo uporabili Levenshteinovo metriko, za ocenjevanje podobnosti med numeričnimi podatki pa evklidsko razdaljo. V tabeli 6.3 so predstavljeni rezultati testiranja.

Z uvedbo mehanizma za ocenjevanje podobnosti med numeričnimi podatki ter pridobivanja podatkov tudi iz sheme smo na ta račun precej izboljšali delovanje v primerjavi z originalno metodo. Naša metoda v primeru delnega prekrivanja podatkov na testnih podatkih in shemah sedaj najde vse preslikave. Poraja pa se vprašanje, kako bi se tako naša kot originalna metoda obnesli v primeru, kadar kompleksna prelikava vsebuje elemente z različnimi podatkovnimi tipi (npr. *Pošta*(varchar) + *PoštnaŠt*(int) ≅ *Pošta*(varchar)). Več o tem v naslednjem poglavju, kjer si bomo ogledali rezultate dodatnih testov.

	$\frac{\text{št. najdenih}}{\text{št. vseh}}$ - natančnost (%)	
Preslikave	**Izboljšana**	**Originalna**
Vse 1:1 preslikave	7/7 - 100	4/7 - 57
Tekstovne preslikave	3/3 - 100	3/3 - 100
Numerične preslikave	4/4 - 100	1/2 - 25
Vse kompleksne preslikave	2/2 - 100	1/2 - 50
Tekstovne preslikave	2/2 - 100	1/2 - 50

Tabela 6.3: Rezultati testiranja – delno prekrivanje podatkov.

Scenarij 2: popolno prekrivanje podatkov

Rezultati drugega dela testiranja nam povedo, kako se metoda obnaša, kadar instanci shem hranita enake podatke. Pričakovati je, da bodo v tem primeru rezultati neprimerno boljši od prvega in tretjega scenarija, ker metoda deluje na podlagi iskanja podvojenih zapisov – več ko jih je, bolj natančne so preslikave, ki jih lahko generira.

V praksi je zelo malo verjetno, da bi naleteli na tak scenarij. Smo pa testiranje na tem scenariju izvajali že od začetka, da smo sproti dobivali informacijo, v kakšnem stanju je naša metoda – če metoda že pri tem scenariju ne dela v redu, potem obstaja velika verjetnost, da tudi pri drugih ne bo.

Tudi v tem primeru smo uporabili entitetno orientirani pristop. Prav tako smo za ocenjevanje podobnosti med podatki uporabili Levenshteinovo in evklidsko razdaljo. Rezultati testiranja so predstavljeni v tabeli 6.4. Opazimo, da tu ne moremo govoriti o izboljšavi, saj v tem scenariju tako originalna kot naša izboljšana metoda najdeta vse preslikave.

Scenarij 3: brez prekrivanja podatkov

To je tretji, najtežji in raziskovalno najbolj zanimiv problem za metode, ki iščejo ujemanja na podlagi podatkov instanc. Ker podvojeni podatki ne obstajajo, metoda z uporabo entitetno orientirane strategije ne more najti

	$\frac{\text{št. najdenih}}{\text{št. vseh}}$ - natančnost (%)	
Preslikave	**Izboljšana**	**Originalna**
Vse 1:1 preslikave	7/7 - 100	7/7 - 100
Tekstovne preslikave	3/3 - 100	3/3 - 100
Numerične preslikave	4/4 - 100	4/4 - 100
Vse kompleksne preslikave	2/2 - 100	2/2 - 100
Tekstovne preslikave	2/2 - 100	2/2 - 100

Tabela 6.4: Rezultati testiranja – popolno prekrivanje podatkov.

preslikav med shemami. Zato se tu uporabi vrednostno orientirani pristop, ki skuša odkriti ujemanja iz konteksta. Glede na originalno metodo pri preslikavah med tekstovnimi elementi pričakujemo nekoliko boljše rezultate, saj naša izboljšana metoda poleg podatkov instanc za iskanje preslikav uporablja tudi podatke sheme (npr. naziv elementov, tip ...).

Na tem mestu velja omeniti, da je vrednostno orientirani pristop za razliko od entitetno orientiranega možno uporabiti v vseh treh scenarijih. Z uporabo vrednostno orientiranega pristopa bi na primerih iz 1. in 2. scenarija dobili primerljive, a mogoče nekoliko manj natančne rezultate, kot če bi uporabili entitetno orientirani pristop.

Teste smo opravili tako na shemah "Real State" in "Inventory" in generiranih shemah. Po opravljenih testih je iz tabele 6.5 razvidno, da smo dosegli 20,75 $\pm$ 1,69 % večjo natančnost. Zasluge za veliko povečanje natančnosti izboljšane metode gre pripisati prav implementaciji iskanja ujemanj med numeričnimi tipi, saj originalna metoda tega ni izvajala. Vidimo, da naša metoda v nekaterih primerih ni našla toliko pravilnih preslikav kot originalna. To je predvsem zato, ker naša metoda poleg podatkov instanc upošteva tudi podatke sheme. Pri testiranju smo ugotovili, da je zelo pomembno, kako nastavimo uteži za obe oceni. Če nastavimo preveliko težo oceni podobnosti iz podatkov sheme, se zgodi, da bodo rezultati slabši, če so imena elementov zelo različna. V našem primeru smo oceni podobnosti instanc podatkov utež

nastavili na $0,7$, oceni podobnosti podatkov sheme pa na $0,3$. Do teh vrednosti smo prišli s poskušanjem. V tem trenutku še nismo preučevali možnosti, da bi algoritem uteži nastavljal samodejno, zato ta problem puščamo odprt in naj služi kot izhodišče za nadaljnje delo.

Velika večina najdenih kompleksnih preslikav sestoji iz elementov, ki predstavljajo demografske podatke (npr. naslovi, imena ...). Največ problemov je s preslikavami numeričnih elementov, posebno takih, ki so v eni shemi v eni merski enoti, v drugi pa v drugačni (npr. *building-area*). Navajamo nekaj primerov najdenih kompleksnih preslikav:

$$\begin{aligned} \textit{ship-address} &\leftrightarrow \texttt{CONCAT(}\textit{ship-address}, \textit{ship-postal-code}, \\ &\qquad \texttt{CONCAT(}\textit{ship-city}, \textit{ship-country}\texttt{))} \\ \textit{house-address} &\leftrightarrow \texttt{CONCAT(}\textit{house-street}, \textit{house-city}, \textit{house-zip-code}\texttt{)} \end{aligned}$$

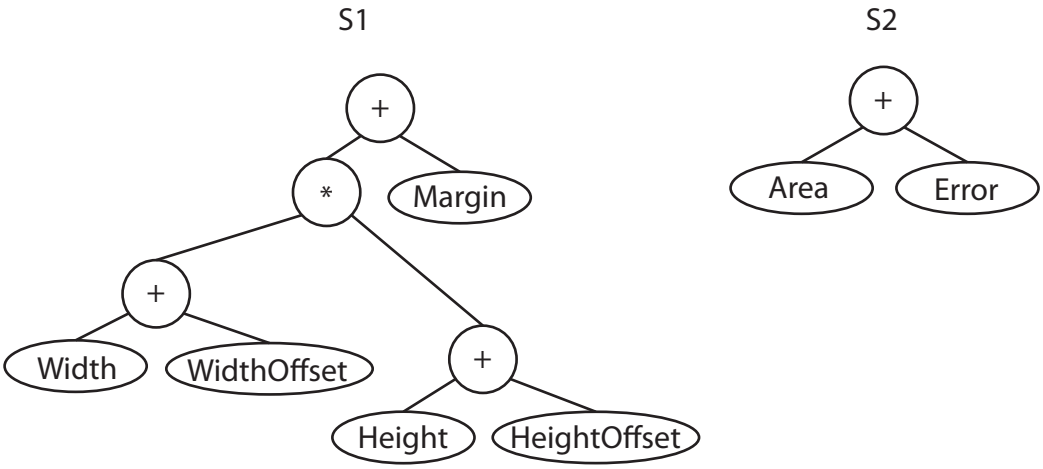
6.2.2 Iskanje kompleksnih preslikav

Poskuse smo izvajali na najbolj enostavnih in pa tudi precej kompleksnih preslikavah. V vseh primerih se je naša metoda dobro odrezala. Na sliki 6.1 je prikazan primer kompleksne numerične preslikave, kakršne je naša izboljšana metoda sposobna odkriti. Problem pa nastane pri kompleksnih preslikavah, kjer se tipi podatkov v preslikavi razlikujejo. Primer take preslikave je prikazan na sliki 6.2. Ker metoda deluje tako, da naenkrat dela samo z enim tipom podatkov, ni možnosti, da bi kot rezultat vrnila popolno preslikavo.[1] To smo poskušali rešiti tako, da bi vse podatke pretvorili v tiste podatkovne tipe, za katere trenutno iščemo preslikave (v tem primeru v tekstovni tip). Na tak način bi preverjali ujemanja z vsemi možnimi elementi in bi lahko dobili pravilne preslikave. Izkazalo se je, da tak način ni zanesljiv. Prav tako lahko ta pristop večinoma uporabljamo samo v situacijah, ko iščemo tekstovne preslikave, saj se ostale podatkovne tipe vsaj na neki način da pretvoriti v tekst, medtem ko na primer teksta ne moremo pretvoriti v števila.

[1] Popolna preslikava je preslikava, v katero so vključeni vsi elementi, ki so resnično del preslikave – brez odvečnih ali manjkajočih elementov.

	$\frac{\text{št. najdenih}}{\text{št. vseh}}$ - natančnost (%)	
Preslikave	**Izboljšana**	**Originalna**
RS Vse 1:1 preslikave	6/7 - 85	6/7 - 85
RS Tekstovne preslikave	5/6 - 84	6/6 - 100
RS Numerične preslikave	1/1 - 100	0/1 - 0
RS Vse kompleksne preslikave	8/12 - 67	3/12 - 25
RS Tekstovne preslikave	4/5 - 80	3/5 - 60
RS Numerične preslikave	4/7 - 57	0/7 - 0
INV Vse 1:1 preslikave	17/27 - 63	11/27 - 40
INV Tekstovne preslikave	8/11 - 72	11/11 - 100
INV Numerične preslikave	9/16 - 56	0/16 - 0
INV Vse kompleksne preslikave	4/11 - 36	2/11 - 18
INV Tekstovne preslikave	2/4 - 50	2/4 - 50
INV Numerične preslikave	2/7 - 28	0/7 - 0
GEN:100 Vse 1:1 preslikave	4/7 - 57	3/7 - 42
GEN:100 Tekstovne preslikave	2/3 - 67	3/3 - 100
GEN:100 Numerične preslikave	2/4 - 50	0/2 - 0
GEN:100 Vse kompleksne preslikave	1/2 - 50	2/2 - 100
GEN:100 Tekstovne preslikave	1/2 - 50	2/2 - 100
GEN:1000 Vse 1:1 preslikave	4/7 - 57	3/7 - 42
GEN:1000 Tekstovne preslikave	2/3 - 67	3/3 - 100
GEN:1000 Numerične preslikave	2/4 - 50	0/2 - 0
GEN:1000 Vse kompleksne preslikave	1/2 - 50	2/2 - 100
GEN:1000 Tekstovne preslikave	1/2 - 50	2/2 - 100

Tabela 6.5: Rezultati testiranja – brez prekrivanja podatkov.

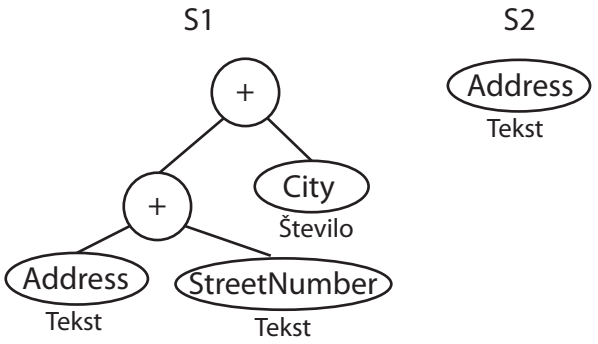


Slika 6.1: Primer kompleksne preslikave numeričnega tipa.

Ena možna rešitev tega bi bila, da se vse podatke predstavi na neki tretji način, kot je na primer vektorska predstavitev. Vsak podatek, tako številski kot tekstovni, bi preslikali v vektorski prostor. Podatke, predstavljene na tak način bi nato lahko primerjali s kosinusno razdaljo, saj je ta metrika ena najprimernejših za računanje podobnosti med vektorji velikih dimenzij. Poleg tega je ta metoda tudi zelo hitra.

6.2.3 Večje število zapisov

Ker je scenarij iskanja ujemanj na podlagi podatkovnih instanc brez medsebojnega prekrivanja najbolj zanimiv in v praksi tudi najbolj pogost, se je porodila ideja, da bi bilo zanimivo testirati, kako se metoda obnese na večjem številu podatkov. Ker imamo za shemi "Real State" in "Inventory" samo 100 zapisov, lahko tu uporabimo samo generirani shemi. V ta namen smo s pomočjo storitve [16] generirali po 1000 zapisov za vsako od shem. Glede na veliko večjo količino podatkov, ki so metodi na voljo za izvajanje primerjav, smo tu pričakovali precejšnje izboljšanje rezultatov v smislu ocene uspešnosti. Če primerjamo grafa na slikah 6.4 in 6.5, vidimo, da temu ni tako. Ocena uspešnosti vseh najdenih preslikav je pri 1000 zapisih v pov-



Slika 6.2: Primer kompleksne preslikave pri kateri se podatkovni tipi ne ujemajo.

prečju komaj 13 % višja kot pri 100 zapisih. Tako se je na račun večjega števila podatkov nekoliko zvišalo zaupanje v najdena ujemanja, samo število pravilno najdenih preslikav pa je v obeh primerih identično (glej tabelo 6.5).

6.2.4 Časovna zahtevnost

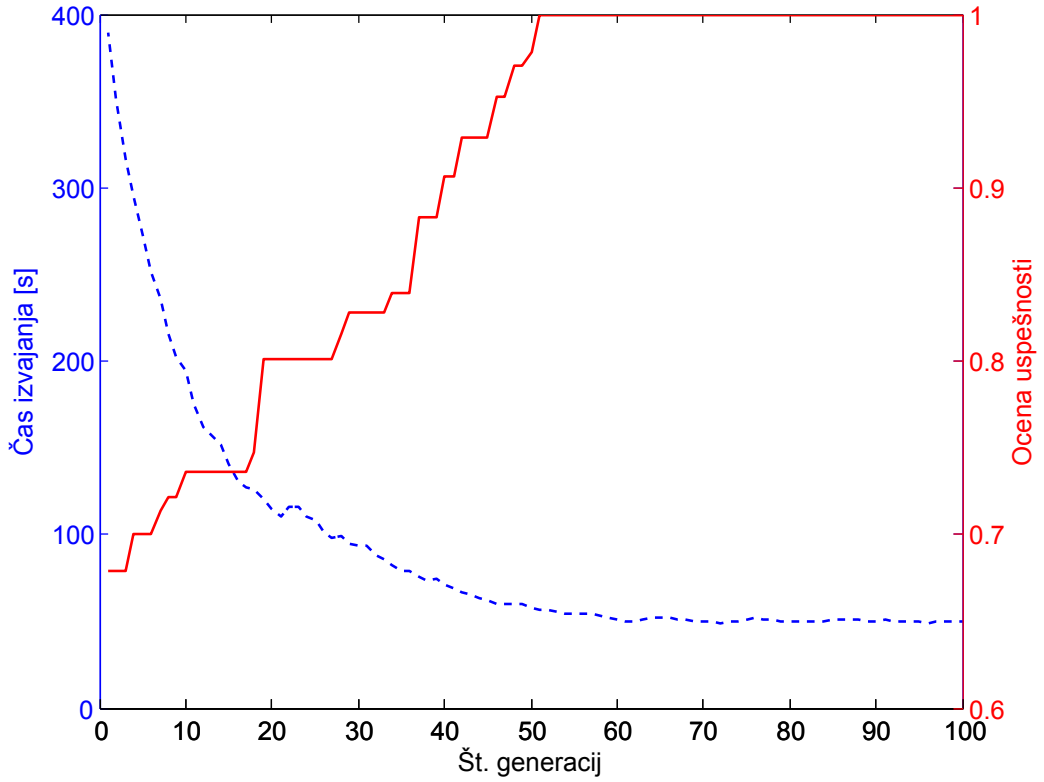
Naša izboljšana metoda ni dodelana do te mere, da bi bila zmožna povsem avtomatsko poiskati vsa možna ujemanja med podanima shemama. To pomeni, da se mora uporabnik za vsako predlagano preslikavo odločiti, ali je preslikava pravilna ali ne. V primeru, da je preslikava pravilna, mora elemente, ki so prisotni v preslikavi, ročno uvrstiti v seznam elementov, ki se jih pri naslednjem iskanju preslikav ignorira. Ker je zato težko izmeriti celoten čas, ki ga potrebuje metoda, da najde vse preslikave, smo se odločili, da bomo merili čas izvajanja prvega zagona metode. Ker metoda ob prvem zagonu izbira iz množice vseh elementov shem, je ta čas zaradi večjega števila možnih kombinacij med njimi praviloma daljši od časa izvajanja vseh naslednjih zagonov. Meritev smo izvedli na shemah in podatkih iz drugega scenarija (delno prekrivanje podatkov). Za tega smo se odločili, ker predstavlja kompromis med prvim in tretjim scenarijem, kar nam omogoča lažje primerjanje obeh pristopov za iskanje ujemanj. Pri testiranju časovne zahtevnosti smo uporabili iste parametre, kot so predstavljeni v tabeli 6.2 v

drugem stolpcu.

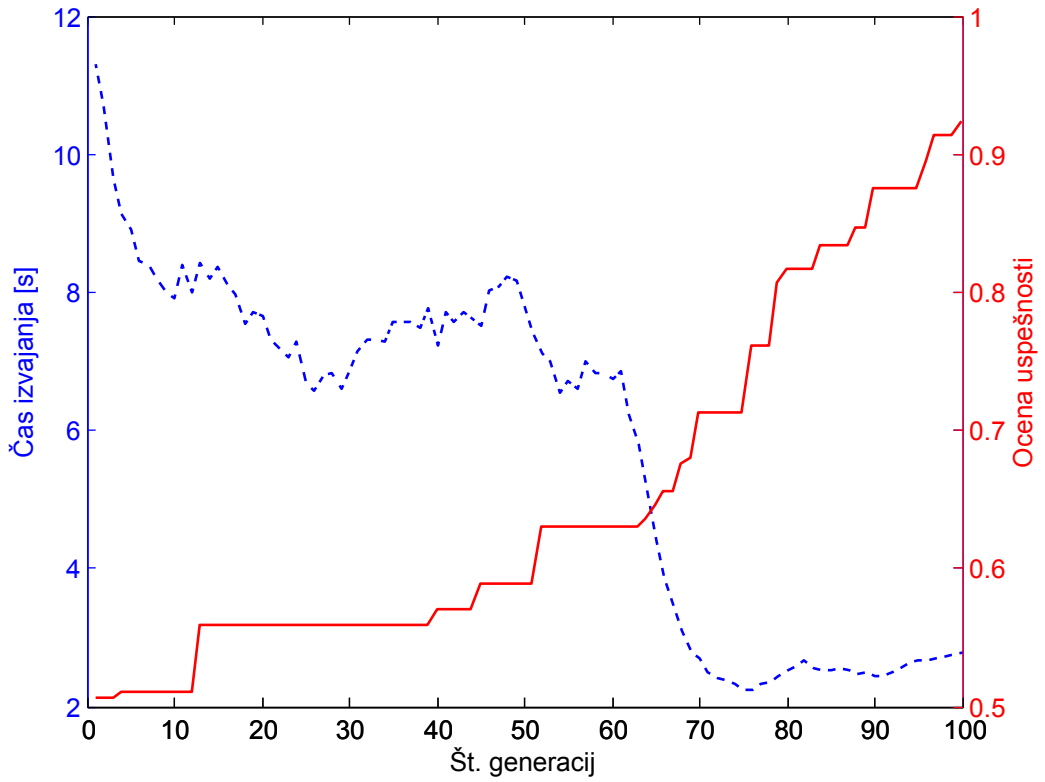
Najprej smo izmerili povprečni čas izvajanja metode, kadar uporabljamo entitetno orientirani pristop za iskanje ujemanj, nato pa smo isto ponovili za vrednostno orientirano strategijo. Iz grafov na slikah 6.3 in 6.4 je razvidno, da je vrednostno orientirana strategija precej hitrejša od entitetno orientirane. Povprečni izvajalni čas metode za izvedbo ene generacije pri uporabi vrednostno orientirane strategije znaša $5,82$ s, medtem ko pri uporabi entitetno orientirane strategije znaša kar $91,82$ s – skoraj 15x več. Razlog za toliko počasnejše izvajanje tiči v različnem načinu izvajanja primerjav med podatki. Pri vrednostno orientirani strategiji se primerjanje med podatki izvaja na celotni množici podatkov hkrati, medtem ko pri entitetno orientirani podobnost računamo za vse možne kombinacije med njimi. Iz tega sledi, da je časovna zahtevnost metode pri uporabi vrednostno orientirane strategije enaka $\mathcal{O}(n)$, pri uporabi entitetno orientirane pa $\mathcal{O}(n^2)$.

Iz grafov opazimo tudi, kako se izvajalni čas manjša v odvisnosti od ocene uspešnosti trenutne populacije. Višja ko je ocena, nižji je izvajalni čas. Razlog za to je, da se preslikave z vsako generacijo čedalje bolj poenostavljajo in konvergirajo k optimalni rešitvi. Bolj ko je preslikava poenostavljena, manj operacij je potrebnih, da evalviramo drevesa, ki sestavljajo preslikavo.

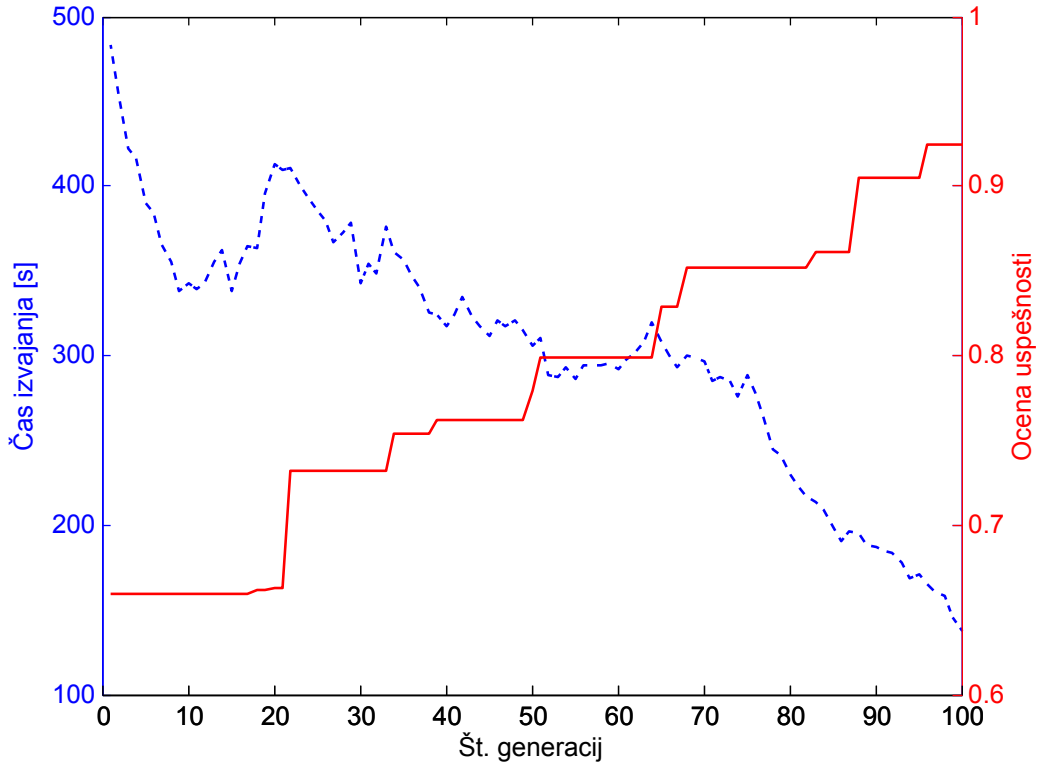
Da bi dobili čase, ki bi bili čimboljši pokazatelj obnašanja naše metode na realnih primerih, smo izmerili še izvajalni čas metode za tretji scenarij z uporabo vrednostno orientiranega pristopa s po 1000 zapisi v instancah posamezne sheme. Rezultati, ki so vidni na sliki 6.5, potrjujejo našo tezo o časovni zahtevnosti iz prejšnjega odstavka, saj kažejo na skoraj linearno povečanje izvajalnega časa v primerjavi z rezultati na sliki 6.4.



Slika 6.3: Čas iskanja ujemanja (modra) in ocena uspešnosti (rdeča) skozi generacije pri uporabi entitetno orientirane strategije.



Slika 6.4: Čas iskanja ujemanja (modra) in ocena uspešnosti (rdeča) skozi generacije pri uporabi vrednostno orientirane strategije.



Slika 6.5: Čas iskanja ujemanja (modra) in ocena uspešnosti (rdeča) skozi generacije pri uporabi vrednostno orientirane strategije pri 1000 zapisih.

Poglavje 7

Sklep

Namen te magistrske naloge je izboljšava ene od že obstoječih metod za iskanje ujemanj med podatkovnimi shemami. Najprej smo pregledali obstoječe pristope na področju iskanja ujemanj med podatkovnimi shemami. Ti pristopi se delijo na tri skupine in se razlikujejo po tem, na podlagi katerih informacij iščejo preslikave. Prvo skupino metod sestavljajo metode, ki ujemanja med podatkovnimi shemami iščejo na podlagi njihovih instanc. V drugo skupino spadajo metode, ki se za iskanje preslikav zanašajo na podatke, pridobljene iz shem – imena in opise elementov, tipe podatkov, omejitve med elementi ... V tretjo skupino pa sodijo hibridne metode, ki združujejo pristope tako iz prve kot druge skupine. Metode hibridne narave so se v praksi izkazale za najbolj učinkovite, saj z uporabo samo enega pristopa le stežka odkrijemo vse možne preslikave.

V nadaljevanju smo podrobno opisali eno od obstoječih metod. Izbrali smo si metodo, ki deluje na principu evoucijskega algoritma in spada v drugo skupino metod – za iskanje preslikav uporablja podatkovne instance. Metoda deluje iterativno, in sicer tako, da v prvem koraku generira naključne možne rešitve (preslikave) danega problema. Preslikave so predstavljene kot pari drevesnih struktur. Vsako drevo je sestavljeno iz listov, ki predstavljajo elemente sheme, in pa notranjih vozlišč, ki predstavljajo možne operacije, ki se lahko izvajajo nad listi oziroma elementi. V vsaki iteraciji se za vsako

preslikavo izračuna ocena uspešnosti, in sicer tako, da drevesa evalviramo na podlagi podatkovnih instanc. V vsaki iteraciji se poleg računanja ocene uspešnosti posameznike še križa in mutira. S temi operacijami dobimo nove posameznike, ki so praviloma boljši kot tisti iz prejšnje iteracije. Na tak način čez nekaj generacij dobimo pravilne preslikave.

Izbrano metodo smo nato predstavili še iz praktičnega vidika. Predstavili smo uporabljene omejitve pri iskanju preslikav, saj bi brez omejitev in predpostavk metoda zaradi preobsežnega prostora možnih rešitev težko našla ustrezne preslikave med shemami. Sproti smo omenili tudi par slabih lastnosti metode, na podlagi katerih smo v nadaljevanju predlagali dve izboljšavi. Prva je uporaba dodatnih informacij, ki so nam na voljo. To je uporaba podatkov, ki jih pridobimo iz sheme – v našem primeru so to nazivi elementov. Podobnost med posameznimi elementi izračunamo z uporabo več funkcij za ocenjevanje podobnosti. S to oceno nato predlagamo potencialne kandidate za preslikave. Z uporabo teh podatkov se izognemo preveliki odvisnosti metode od podatkovnih instanc, prav tako pa lahko na ta način povečamo njeno natančnost. Druga predlagana izboljšava pa je razširitev iskanja ujemanj še na druge tipe podatkov. Ker izbrana metoda deluje samo s tekstovnimi podatki, smo se odločili, da podpremo še numerične tipe.

Za konec smo še primerjali našo izboljšano metodo z originalno metodo. To smo izvedli s pomočjo testnih shem in podatkov, ki so jih uporabljali avtorji za testiranje originalne metode. Testiranje smo izvedli na treh možnih scenarijih, ki se lahko zgodijo glede na podvajanje podatkov v instancah: delno prekrivanje, popolno prekrivanje in brez prekrivanja podatkov. Teste za prvi in drugi scenarij smo izvedli na naključno generiranih shemah in podatkih, za tretji scenarij pa na shemah in podatkih, ki so dostopni na [14]. Vsak scenarij smo testirali na 100 zapisih. Odkrili smo, da naša metoda v vseh treh scenarijih daje boljše rezultate kot originalna. Največje izboljšanje natančnosti v primerjavi z originalno metodo smo dosegli v tretjem scenariju, kjer se pokaže odvisnost originalne metode od podatkovnih instanc. Ker naša metoda upošteva tudi imena elementov, daje boljše rezultate. V primerjavi z

originalno metodo je naša izboljšana metoda na vseh treh scenarijih za 22,42 ± 2,39 % bolj natančna.

Poleg same primerjave z originalno metodo smo izvedli še nekaj dodatnih testov. Generiranim shemam smo dodali nekaj lastnih kompleksnih preslikav, da bi otežili postopek iskanja ujemanj. Ugotovili smo, da izboljšana metoda uspešno najde tudi bolj kompleksne preslikave. Naleteli pa smo na težavo, ki se pokaže, kadar iščemo kompleksne preslikave med elementi različnih podatkovnih tipov. Ker naša metoda naenkrat išče ujemanja samo za točno določen podatkovni tip elementov, takšnih preslikav ni sposobna odkriti.

V praksi se redko srečamo s podatkovnimi instancami, ki bi vsebovale samo 100 zapisov. Zato smo opravili test, kako se metoda obnaša, kadar je na voljo 1000 zapisov. Pred izvedbo testa smo postavili tezo, da bo metoda zaradi večjega števila podatkov našla več preslikav. Izkazalo se je, da ni tako. Metoda je v tem primeru našla identične preslikave kot pri testu s 100 zapisi, vendar s to razliko, da se je povečalo zaupanje v najdene preslikave. Skupaj s testiranjem natančnosti metode, kadar ima opravka s 100 in 1000 zapisi v podatkovnih instancah, smo merili tudi izvajalni čas metode. Ugotovili smo, da je metoda z uporabo vrednostno orientirane strategije neprimerno hitrejša kot z uporabo entiteno orientirano. Z meritvami pa smo pokazali tudi, da izvajalni čas narašča linearno z velikostjo podatkovnih instanc.

Možnosti za nadaljnje delo in izboljšave naše metode je ogromno. Glavna izboljšava, ki bi močno izboljšala delovanje naše metode, je implementacija pristopa za ocenjevanje podobnosti med različnimi tipi podatkov. Ker trenutno naša metoda išče ujemanja samo med elementi enega tipa podatkov, na tak način ne more uspešno odkriti vseh možnih preslikav – predvsem takšnih, ki vsebujejo elemente različnih podatkovnih tipov. Druga pomembna nadgradnja bi bila vpeljava nove metrike za ocenjevanje podobnosti med podatki, ki bi bila sposobna primerjati podatke, ki so predstavljeni na različne načine (npr. različni formati datumov, kategorizirani podatki ...). Ker pa je pri iskanju preslikav zelo pomemben tudi čas iskanja preslikav, bi bilo treba v nadaljevanju nekaj dela nameniti optimizaciji same metode, saj smo na podlagi

meritev videli, da pri večjih količinah podatkov izvajalni čas precej naraste.

Literatura

[1] M. G. de Carvalho, A. H. F. Laender, M. A. Goncalves, A. S. da Silva, "An evolutionary approach to complex schema matching", *Information Systems*, št. 3, zv. 38, str. 302–316, 2013.

[2] M. G. de Carvalho, A. H. F. Laender, M. A. Goncalves, A. S. da Silva, "A genetic programming approach to record deduplication", *Transactions on Knowledge and Data Engineering*, št. 3, zv. 24, str. 399–412, 2012.

[3] E. Rahm, P. A. Bernstein, "A survey of approaches to automatic schema matching", *The VLDB journal 10*, str. 334–350, 2001.

[4] P. A. Bernstein, J. Madhavan, E. Rahm, "Generic schema matching, ten years later", *Proceedings of the VLDB Endowment*, št. 11, zv. 4, str. 695–701, 2011.

[5] J. Madhavan, P. A. Bernstein, E. Rahm, "Generic schema matching with cupid", *VLDB*, zv. 1, str. 49–58, 2001.

[6] P. Shvaiko, J. Euzenat, "A survey of schema-based matching approaches", *Journal on Data Semantics IV*, str. 146–171, 2005.

[7] H. Do, E. Rahm, "COMA: a system for flexible combination of schema matching approaches", *Proceedings of the 28th international conference on Very Large Data Bases*, str. 610–621, 2002.

[8] R. Russel, M. Odell, "Soundex", *US Patent*, št. 1, 1918.

[9] P. Lawrence, "The double metaphone search algorithm", *C/C++ users journal*, št. 6, zv. 18, str. 38–43, 2000.

[10] S. Agrawal, S. Chaudhuri, G. Das, A. Gionis, "Automated ranking of database query results", *In CIDR*, 2003.

[11] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, P. Domingos, "iMAP: discovering complex semantic matches between database schemas", *Proceedings of the 2004 ACM SIGMOD international conference on management of data*, str. 383–394, 2004.

[12] A. Algergawy, R. Nayak, G. Saake, "Element similarity measures in XML schema matching", *Information Sciences*, št. 24, zv. 180, str. 4975–4998, 2010.

[13] A. Z. Gazvoda, "Integracija podatkovnih shem na osnovi analize podatkov z algoritmom arhetipske analize za povzemanje podatkovnih množic", *magistrska naloga*, 2014

[14] Zbirka shem namenjenih testiranju iskanja ujemanj. Dostopno na: `http://pages.cs.wisc.edu/~anhai/wisc-si-archive/summary.name.html` (pridobljeno 19. 11. 2014).

[15] Programski paket Febrl. Dostopno na: `http://datamining.anu.edu.au/projects/linkage.html` (pridobljeno 19. 11. 2014).

[16] Storitev GenerateData. Dostopno na: `http://http://www.generatedata.com/` (pridobljeno 23. 1. 2015).

[17] S. Melnik, H. Garcia-Molina, E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching", *Data Engineering*, str. 117–128, 2002.

[18] M. Mitchell, *An introduction to genetic algorithms*, MIT press, 1998.

[19] D. Beasley, R. R. Martin, D. R. Bull, "An overview of genetic algorithms: Part 1. Fundamentals", *University computing*, zv. 15, str. 58–69, 1993.

[20] G. Jones, "Genetic and evolutionary algorithms", *Encyclopedia of computational chemistry*, str. 323–330, 1998.

[21] K. Sastry, D. Goldberg, G. Kendall, "Genetic algorithms", *Search methodologies*, str. 97–125, 2005.

[22] W. E. Winkler, "The state of record linkage and current research problems", *Statistical research division*, 1999.

[23] W. Cohen, P. Ravikumar, S. Fienbarg, "A comparison of string metrics for matching names and records", *Kdd workshop on data cleaning and object consolidation*, zv. 3, str. 73–78, 2003.

[24] A. M. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida", *Journal of the American Statistical Association*, št. 406, zv. 84, str. 414–420, 1989.

[25] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", *Soviet physics doklady*, zv. 10, št. 8, str. 707–710, 1966.

[26] H. H. Do, S. Melnik, E. Rahm, "Comparison of schema matching evaluations", *Web, web-services, and database systems*, str. 221–237, 2003.

[27] Z. Huang, "Clustering large data sets with mixed numeric and categorical values", *Proceedings of the 1st Pacific-Asia conference on knowledge discovery and data mining*, str. 21–34, 1997.

[28] H. H. Do, "Schema matching and mapping-based data integration", *doktorska disertacija*, 2006.

[29] J. Evermann, "Theories of meaning in schema matching: an exploratory study", *Information Systems*, zv. 34, št. 1, str. 28–44, 2009.

[30] Z. Bellahsene, A. Bonifat, E. Rahm, *Schema matching and mapping*, zv. 20, 2011.

[31] A. Bilke, F. Naumann, "Schema matching using duplicates", *Proceedings of the 21st international conference on data engineering*, str. 69–80, 2005.